Universität Potsdam
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Physik und Astronomie

Die Rekonstruktion von Luminositäts- und Radiusfunktion von Gammaquellen mithilfe von Machine-Learning Algorithmen

Arbeit zur Erlangung des akademischen Grades
"Bachelor of Science"
in der Wissenschaftsdisziplin Physik

Eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

Von
Benjamin Biewald

1. Gutachterin: Dr. Kathrin Egberts
2. Gutachter: Prof. Dr. Christian Stegmann

29. März 2021

Kurzfassung

Gammaquellen sind Objekte (u.a. einige Supernova-Überreste, Pulsarwindnebel oder binäre Sternsysteme), die γ-rays emittieren. Durch Wechselwirkung von relativistischen Teilchen, welche zum Teil in jenen Quellen produziert und beschleunigt werden, mit ausgedehnten Magnetfeldern und anderen Teilchen, werden hochenergetische Gammaquanten erzeugt, die durch bodengebundene Cherenkov-Teleskope über Teilchenschauer detektiert werden können. Mithilfe des Katalogs des H.E.S.S. Galactic Plane Surveys (HGPS) wurden in dieser Arbeit die Verteilungen von Luminosität und Radius von galaktischen Gammaquellen unter der Annahme, dass diese Eigenschaften unabhängig voneinander potenzverteilt sind, rekonstruiert. Um dem Beobachtungsbias der HGPS-Daten zu begegnen, wurden künstliche Gammaquellen simuliert und aus diesen, unter Berücksichtigung aller auch im HGPS auftretenden Ungenauigkeiten, Datasets berechnet. Dabei wurden vier Modelle zur Beschreibung der räumlichen Verteilung der Gammaquellen verwendet.
Mit diesen konnten Convolutional Neural Networks auf die Rekonstruktion der Modellparameter α_L und α_R trainiert und anschließend evaluiert werden. Es ergab sich, dass die stochastische Abschätzung unbekannter Informationen von Quellen, wie die Entfernung oder eine nicht messbare Ausdehnung, verlässlichere Vorhersagen zur Folge hatte, als wenn man nur einen kleineren Teil an Quellen bekannter Entfernung und Ausdehnung berücksichtigte. Die rekonstruierten Parameter lagen zwischen $-2,5$ und $-2,0$ für die Luminosität, und zwischen $-0,3$ und $1,0$ für den Radius. Verschiedene Modellannahmen, wie die Potenzverteilung von Luminosität und Radius, könnten allerdings zu einfach für die realen Verhältnisse sein.

Inhaltsverzeichnis

1 Galaktische Gammaquellen

1.1 Kosmische Strahlung und γ-rays

Um zu verstehen was Gammaquellen (bzw. γ-Quellen) sind, muss man einen Blick auf das nicht-thermische Universum und insbesonders auf kosmische Strahlung werfen. Diese Teilchenstrahlung besteht aus hochenergetischen, geladenen Teilchen, vor allem aus Protonen und schwereren Atomkernen [1]. Diese primäre Strahlung findet ihren Ursprung in nicht-thermischen Prozessen, da selbst das Planck-Spektrum der heißesten Objekte (z.B. Akkretionsscheiben) geringe Energien nur bis zu $10^4\,\mathrm{eV}$ erreicht [2].
Damit eben jene hohe Energien enstehen können, müssen die geladenen Teilchen erst durch bestimmte Beschleunigungsmechanismen an Energie gewinnen. Einen möglichen Mechanismus stellt dabei die sogenannte Fermi-Beschleunigung erster Ordnung dar. Dabei treffen relativistische Teilchen auf eine Schockfront (z.B. um einen Supernova-Überrest) und durchlaufen diese mehrmals aufgrund von Turbulenzen hinter und Unregelmäßigkeiten vor der Front. Mit jedem Durchlauf gewinnt das Teilchen an Energie bis es der Schockfront entkommt und über bestimmte Mechanismen, welche im Folgenden erklärt werden, Gammaquanten erzeugt [2–4].
Die geladenen Teilchen, die bei der Erzeugung der Gammastrahlung eine Rolle spielen, wechselwirken mit irregulären, ausgedehneten Magnetfeldern und werden dadurch gestreut. Dabei erzeugen sie auch elektromagnetische Strahlung. Der Ursprung der primären Teilchen lässt sich kaum nachverfolgen. Die erzeugte elektromagnetische Strahlung erreicht uns aber als diffuse Gammastrahlung und liefert uns Informationen über die Fortbewegungsmechanismen primärer kosmischer Strahlung in galaktischen Magnetfeldern [5]. Erst ab Teilchenenergien von $> 10^{18}\,\mathrm{eV}$ ist der Einfluss der Magnetfelder nicht mehr signifikant, doch der Fluss an Teilchen mit derart hohen Energien ist sehr gering [1].
Aber nicht nur durch Wechselwirkung mit Magnetfeldern, sondern auch durch Wechselwirkung mit Materie können Photonen produziert werden. Wenn die primären Teilchen mit interstellarer Materie wechselwirken, werden dadurch sekundäre Teilchen wie Kaonen und Pionen erzeugt, welche weiter in neutrale Botenteilchen wie Neutrinos oder eben auch hochenergetische Photonen zerfallen. Diese können sich nahezu ungehindert auf geradlinigen Bahnen ausbreiten und liefern so auch Informationen zu ihrem Ursprung [1]. Obwohl die kosmische Strahlung in ihrer Fortbewegung durch Magnetfelder abgelenkt wird, kann man die durch Wechselwirkungen entstandene elektromagnetische Strahlung zurückverfolgen. Während diffuse Emissionen Aufschluss über die Ausbreitung kosmischer Strahlung geben, sind Gammaquellen damit auch Informationsträger für Quellen kosmischer Strahlung.
Folgende Produktionsmechanismen spielen bei der Erzeugung von γ-rays eine Rolle.

Bremsstrahlung
Werden geladene Teilchen in Coloumb-Feldern anderer Ladungen abgelenkt, emittieren sie Bremsstrahlungsphotonen. Im Gegensatz zur Synchrotronstrahlung (bei Ablenkung geladener Teilchen in Magnetfeldern) liefert die Bremsstrahlung energetisch höhere Beiträge im Energiespektrum einer Quelle (ungefähr zwischen $10^6\,\mathrm{eV}$ und $10^{13}\,\mathrm{eV}$) [1].

Der inverse Compton-Effekt
Durch Kollisionen zwischen Photonen und Elektronen kann sowohl Energie der Photonen an die Elektronen übertragen werden (Compton-Effekt), als auch umgekehrt (inverser Compton-Effekt). Da durch die kosmische Hintergrundstrahlung oder durch Sternenlicht zahlreiche Photonen in der Nähe möglicher Quellen vorhanden sind, können wegen dieses Effektes einige durch Kollisionen mit hochenergetischen Teilchen an Energie dazugewinnen [1].

Der Zerfall neutraler Pionen (π^0-Zerfall)
Bei Wechselwirkung zwischen relativistischen Protonen und Kernen mit Materie, können Kaonen und Pionen entstehen. Während geladene Pionen in Myonen und Neutrinos zerfallen, zerfallen neutralen Pionen wiederum sehr schnell in zwei Gammaquanten [1–3]. In diesem Fall ist die Produktion der Strahlung nicht mehr leptonischen, sondern hadronischen Ursprungs.

Neben diesen Prozessen existieren auch andere, wie Materie-Antimaterie-Vernichtung, Kernumwandlungen oder Cherenkov-Strahlung, welche aber Photonen geringerer Energien (z.B. im optischen Bereich) erzeugen [1]. Daher sind sie hier nicht von Relevanz.
Gammaquellen sind nun jene Objekte, welche diese elektromagnetische Strahlung mit hohen Energien ($> 10^5\,\mathrm{eV}$ bis teilweise sogar $> 10^{13}\,\mathrm{eV}$) emittieren. Dabei gilt Strahlung mit einer Energie bis zu etwa $15 \times 10^6\,\mathrm{eV}$ als Soft γ-ray, höhere Energien als High-Energy (HE) γ-rays und Energien ab ungefähr $10^{11}\,\mathrm{eV}$ sogar als Very-High-Energy (VHE) γ-rays [2]. Folgende Objekte wurden bereits im Bereich der Gammastrahlung beobachtet.

Supernovae und deren Überreste
Wegen der gewaltigen Energiefreisetzung und der Existenz einer Schockfront (und der damit einhergehenden Beschreibung von Beschleunigungsmechanismen kosmischer Strahlung) gelten Supernovae als vielversprechende Kandidaten für Gammaquellen [2, 4]. So wurden auch schon verschiedene Supernovaüberreste, wie RX J1713.7-3946, als Gammaquellen identifiziert.

Pulsare und Pulsarnebel
Pulsare sind schnell rotierende Neutronensterne, deren ausgestoßener Pulsarwind ebenfalls eine Schockregion erzeugt. Zusätzlich wirken die starken elektrischen Felder und Magnetfelder als Beschleuniger für geladene Teilchen [2, 6]. Der berühmteste und leuchtkräftigste Vertreter dieser Klasse ist der Krebsnebel, dessen Strahlungsfluss oft auch als Referenz für andere Quellen verwendet wird.

Binäre Sternsysteme in denen Akkretion stattfindet
Unter bestimmten Umständen kann in den Akkretionsscheiben um Neutronensterne oder schwarzen Löchern in diesen Systemen, neben der ohnehin schon vorhandenen thermischen Röntgenstrahlung, durch den inversen Compton-Effekt auch Gammastrahlung entstehen [2].

Außer diesen Objekten kommen auch z.B. materieansammelnde schwarze Löcher in aktiven Galaxienkernen als extragalaktische Gammaquellen in Frage [1]. Diese Arbeit behandelt allerdings nur galaktische Gammaquellen. Unter diesen gibt es aber auch Objekte, wie HESS J1507-622, bei denen eine Zuteilung in eine bestimmte Gruppe bisher nicht eindeutig erfolgen konnte.

1.2 H.E.S.S. Galactic Plane Survey

Gerade weil Gammastrahlung nicht nur über verschiedene Phänomene innerhalb und außerhalb unserer Galaxis entsteht und sich nahezu ungehindert auf geraden Linien ausbreiten kann, sondern sich auch durch bodengebundene Astronomie nachweisen lässt, spielt sie eine wichtige Rolle bei der Erfoschung des nicht-thermischen Universums [5].
Eben jene bodengebundene Detektion galaktischer Gammaquellen findet, unter anderem, im Rahmen des H.E.S.S.-Experiments statt. Der Name „H.E.S.S." ist ein Akronym für „High Energy Stereoscopic System" in Anlehnung an den Entdecker der kosmischen Strahlung Victor Hess, der dafür 1936 mit dem Nobelpreis für Physik ausgezeichnet wurde. Das H.E.S.S.-Experiment ist ein in Namibia ansässiges Experiment bestehend aus mehreren Cherenkov-Teleskopen, welches sich dem Ziel verschrieben hat die experimentelle Basis zur Erforschung der Beschleunigungsmechanismen, Fortbewegung und Interaktionen nicht-thermischer Teilchen zu liefern. Dabei decken die Teleskope einen Energiebereich von $10^{10}\,\mathrm{eV}$ bis $10^{14}\,\mathrm{eV}$ ab [7].

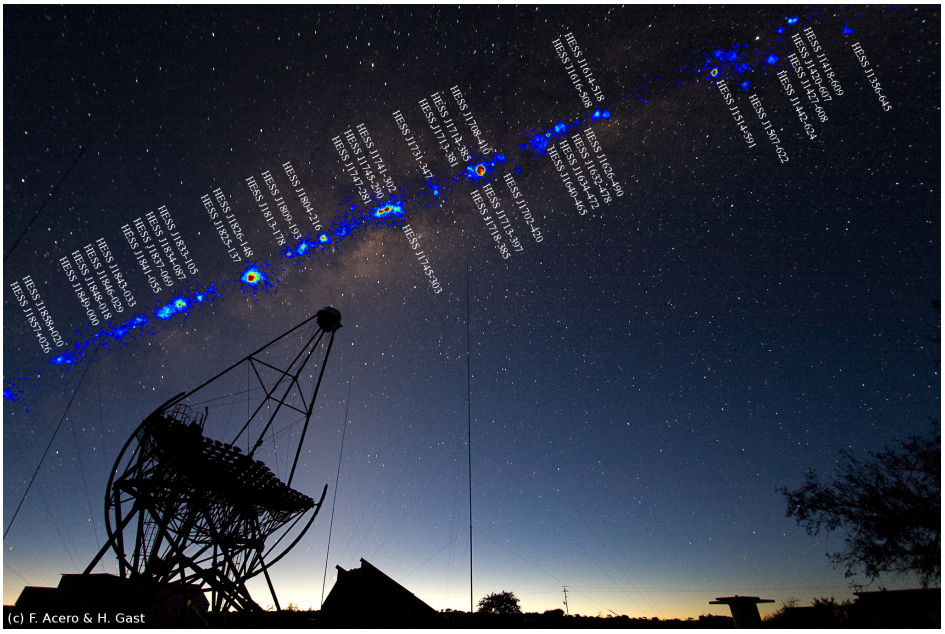


Abbildung 1.1: Eine Montage der von HESS gefundenen Gammaquellen auf ein Bild der Milchstraße mit Cherenkov-Teleskop im Vordergrund, Abbildung von F. Acero und H. Gast, unverändert, veröffentlicht unter der CC-Lizenz siehe `https://creativecommons.org/licenses/by-sa/4.0/deed.de`, letzter Zugriff am 27.03.2021

Wie der Name schon vermuten lässt, detektieren Cherenkov-Teleskope (IACT, Imaging Atmospheric Cherenkov Telescopes) die schwache, optische Cherenkov-Strahlung, welche relativistische Teilchen in der Erdatmosphäre emittieren, da sich diese mit einer höheren Geschwindigkeit als der jeweiligen Lichtgeschwindigkeit in der optisch dickeren Luftschicht bewegen. Treffen HE bzw. VHE γ-rays auf die obere Erdamtosphäre erzeugen sie durch Wechselwirkung mit den Atomen in der Luft einen Teilchenschauer, also eine Kaskade an sekundären Teil-

chen, welche wiederum in neue Teilchen zerfallen bzw. neue Teilchen erzeugen. Diese Teilchen umgibt nun ein Lichtkegel aus Cherenkov-Strahlung, welcher aus Richtung des Teilchenschauers kommt und auf den Erdboden trifft. Stehen dort IACTs, reflektieren deren Spiegel die schwache Strahlung auf einen schnellen Photo-Detektor. Dieser kann nun den wenige Nanosekunden andauernden Lichtblitz detektieren. Haben mehrere Teleskope das Licht wahrnehmen können, kann durch stereoskopische Rekonstrukton nun die Form des Teilchenschauers und damit der Ursprung der Gammastrahlung ermittelt werden. Die Intensität des gemessenen Cherenkov-Lichts gibt Aufschluss über die Energie des Gammaquanten [2]. Abbildung 1.1 zeigt exemplarisch ein solches Cherenkov-Teleskop, während im Hintergrund von H.E.S.S. detektierte Gammaquellen auf ein Bild der Milchstraße montiert wurden.
Von 2004 bis 2013 wurde der H.E.S.S Galactic Plane Survey (HGPS) durchgeführt, in dem mit nahezu 2700 Stunden die Milchstraße in einem Bereich von 250° bis 65° galaktischer Länge und ±3° galaktischer Breite, beobachtet und 78 Gammaquellen entdeckt und kategorisiert wurden. Das Ergebnis ist neben einer galaktischen Karte (Abbildung 1.2), die die Regionen der Beobachtungen zeigt, sowie deren jeweiligen Fluss an VHE γ-rays und die zugehörige Beobachtungszeit veranschaulicht, auch der HGPS Katalog mit insgesamt 78 Gammaquellen. Dabei konnten 16 neue Quellen entdeckt werden und insgesamt 31 Quellen als Pulsarwindnebel, Supernova Überrest, binäres Sternsystem o.ä. identifiziert werden [7].

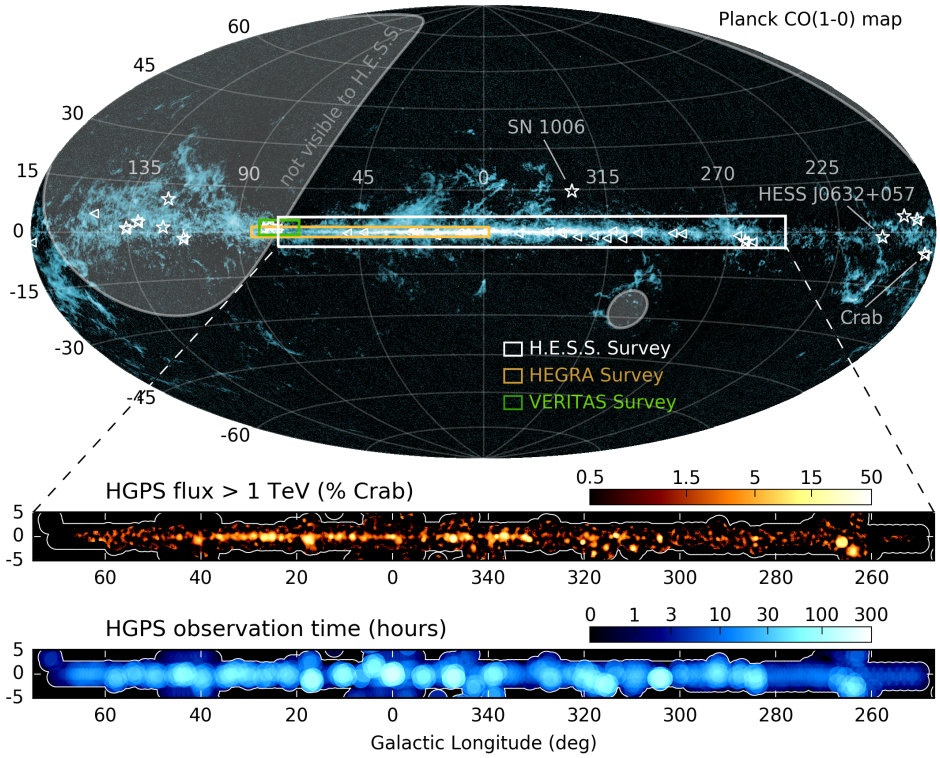


Abbildung 1.2: Galaktische Karte des HGPS; oben: der weiße Kasten zeigt grob das Gesichtsfeld des HGPS im Vergleich zum HEGRA Survey (braun) und dem VERITAS Survey (grün), grau schattiert sind Bereiche, die von H.E.S.S. nicht beobachtet werden können; unten: für das Gesichtsfeld vom HGPS sind der jeweilige Strahlungsfluss > 1 TeV in Einheiten des Strahlungsflusses des Krebsnebels und die jeweilige Beobachtungszeit in Stunden dargestellt, die weißen Konturen zeigen dabei detaillierter den Rand des Gesichtsfelds vom HGPS, Abbildung aus: [7]

Die vorliegenden Katalog-Daten können nun genutzt werden um die Verteilung von Luminosität und Radius der Quellen zu rekonstruieren. Da allerdings von nur ungefähr einem Drittel aller Quellen die Entfernung abschätzbar ist, sowie die mittlere Auflösung des HGPS bei 0.08° liegt und damit alle Quellen, welche kleiner erscheinen, nur als Punktquellen wahrgenommen werden, bleiben insgesamt nur 16 Quellen von denen man die Entfernung und die Ausdehnung kennt. Diese Informationen sind unablässlich, um aus dem gemessenen Strahlungsfluss die Luminosität der Quelle, bzw. aus der gemessenen Ausdehnung den Radius der Quelle zu berechnen. Eine weitere Einschränkung ist der Beobachtungsbias, der darin besteht, dass man vorallem helle Quellen detektiert. Leuchtstarke bzw. nahe Quellen können eher detektiert werden als leuchtschwache und/oder weit entfernte Gammaquellen, womit man keine repräsentative Schnittmenge an gefundenen Gammaquellen erhält. Ein Ansatz, um diesem Bias zu begegnen, stellt die Simulation künstlicher Gammaquell-Populationen dar, aus denen, unter Berücksichtigung aller Beobachtungseffekte, ein Datensatz berechnet wird. Mithilfe von Machine-Learning sollen aus diesem mehr Informationen gewonnen werden, so dass eine realistische Rekonstruktion der Verteilungsfunktion von Luminosität und Radius gelingt.

2 Machine-Learning

2.1 Neuronale Netze

Unter maschinellem Lernen bzw. Machine-Learning versteht man Algorithmen, welche selbstständig Zusammenhänge in vorgegebenen Daten erlernen und so unter vergleichweise geringem Aufwand komplexe Aufgaben lösen können (z.B. Klassifikationen). In dieser Arbeit dient Machine-Learning einer Regressionsanalyse ($f : \mathbb{R}^n \to \mathbb{R}^m$) um eine Beziehung zwischen (auch mehrdimensionalen) Input und Output zu modellieren. Wenn die Outputs mit zu erwartenden Ergebnissen verglichen werden können, spricht man hier auch von überwachtem Lernen oder supervised Learning [8].
Die Berechnung des Outputs erfolgt über ein sogenanntes neuronales Netzwerk, also ein Verbund mehrerer Knotenpunkte (genannt Neuronen in Anlehnung an biologische neuronale Netze), deren Verbindungen untereinander unterschiedlich stark gewichtet sind. Es gibt verschiedene mögliche Architekturen für neuronale Netze, meist sind die Neuronen aber in mehreren Schichten, genannt Layer, angeordnet. Abbildung 2.1 zeigt schematisch ein sogenanntes Feedforward Netzwerk (FFN) in dem die Input Daten (links) an ein sog. Hidden Layer mit drei Neuronen (mittig) übergeben und schließlich in einem Output Layer (rechts) ausgegeben werden. Die Zahlen an den Pfeilen stellen die unterschiedlichen Gewichtungen der Verbindung dar, während die Zahlen in den Kreisen Ergebnisse der Berechnungen innerhalb des jeweiligen Neurons darstellen. Die Berechnung erfolgt zuerst linear (die jeweils obere Zahl pro Neuron) und anschließend nicht-linear (die darunter liegende Zahl) über eine sogenannte Aktivierungsfunktion. Wie dies konkret aussieht, folgt im nächsten Unterabschnitt. Da die Gewichtungen anfangs zufällig festgelegt werden, sind auch die ersten Outputs zufällig und entsprechen meist nicht dem gewünschten Ergebnis („calculated“ und „target“ in Abb. 2.1).

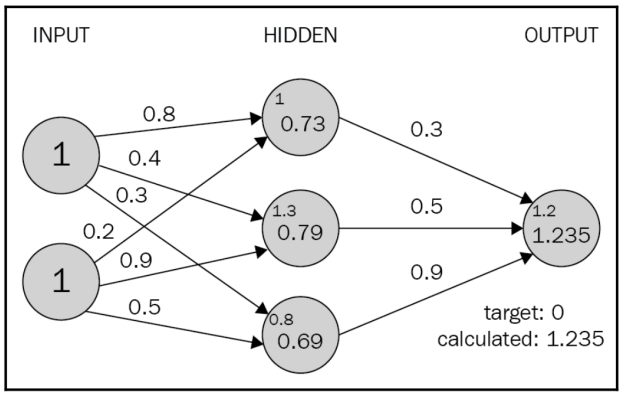


Abbildung 2.1: Aufbau eines Feedforward Netzes (FFN), die Daten durchlaufen das Netz von links nach rechts. Die Zahlen an den Pfeilen stellen unterschiedliche Gewichtungen der Verbindungen dar und die Zahlen in den Kreisen sind die Outputs der Neuronen vor und nach dem Durchlaufen der Aktivierungsfunktion, Grafik aus [9]

Sind alle Neuronen eines Layers mit allen des darauffolgenden verknüpft, spricht man von einem fully-connected (fc) Layer. Die Zahl der Layer und der Neuronen in den Layern definieren die sog. Tiefe und Breite des Netzes. Diese können, je nach Problemstellung, beliebig varriert werden. Verwendet ein neuronales Netz mehrere Hidden Layer, nennt man dies deswegen auch Deep-Learning, da das Netzwerk besonders „tief“ ist [10].
Oft werden zur Analyse zweidimensionaler Daten, anstelle von FFNs, Netzwerke mit anderen Layern wie Convolutional Neural Networks (CNN) verwendet. Abbildung 2.2 zeigt den schematischen Aufbau eines solchen Netzes. Hier erfolgen die Berechnungen in den Neuronen über Faltung (englisch: Convolution), was im Falle von zweidimensionalen Inputs (wie z.B. Bildern) durch einen geringeren Rechenaufwand schneller erfolgt. Dadurch sind die Vorhersagen meist auch zuverlässiger. Zwischen den Convolutional Layern (in der Abbildung *conv* genannt) sorgen Pooling Layer (*pool*) dafür, die Inputs zu verkleinern ohne dabei wichtige Informationen zu verlieren. Convolutional Layer und Fully Connected Layer lassen sich auch kombinieren, wenn man, wie in dieser Arbeit, aus zweidimensionalen Inputs einen eindimensionalen Output erlangen will.

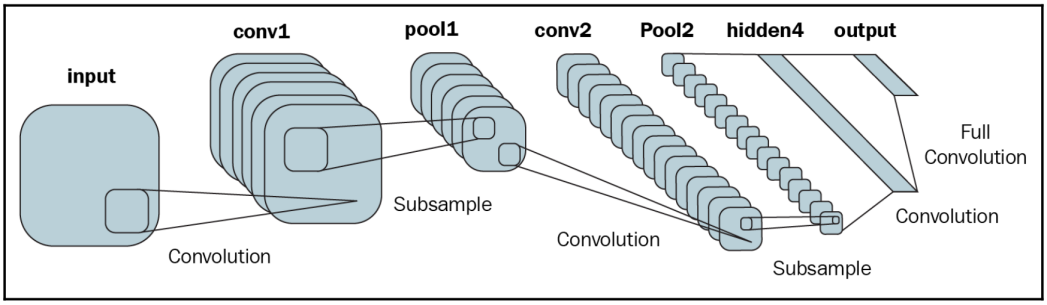


Abbildung 2.2: Aufbau eines Convolutional Neural Networks (CNN), die Daten durchlaufen das Netz von links nach rechts. Convolutional Layer extrahieren Informationen aus den (hier zwei dimensionalen) Inputs und Pooling Layer komprimieren die Daten, Grafik aus [9]

Die Funktionsweise der Pooling-Layer zeigt Abbildung 2.3 schematisch. Eine Matrix (der *Kernel*) fährt die zweidimensionalen Daten Stück für Stück, vorgegeben durch eine Schrittweite, ab und berechnet die Elemente einer kleineren Version der ursprünglichen Matrix. In der Grafik 2.3 ist dieser Prozess mit einem 3x3 großen Kernel und einer Schrittweite von 2 dargestellt. Je nach Art des Pooling wird der Mittelwert aller vom Kernel abgebildeten Elemente gebildet (Average-Pooling), oder aber auch der höchste Wert übernommen (Max-Pooling).

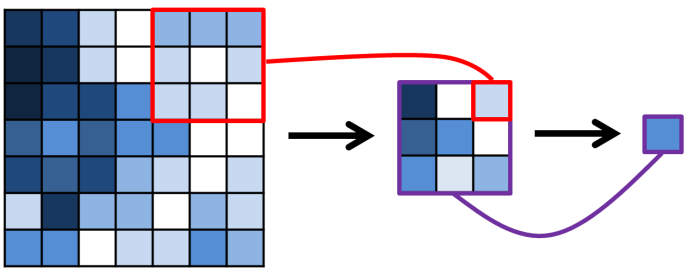


Abbildung 2.3: Schematische Darstellung des Average-Poolings mit einem 3x3 Kernel (Eigene Grafik)

2.2 Forward- und Backward-Propagation

Der eigentliche Lernprozess besteht nun aus zwei Phasen, welche mehrmals wiederholt werden. Eine solche Iteration nennt man Epoche.
Die erste Phase ist die Forward-Propagation, in der die Daten das Netzwerk vom Input bis zum Output Layer durchlaufen und die Berechnungen der Outputs des jeweiligen Neurons stattfinden. In einem fc-Layer erfolgt die Berechnung des Outputs h_j des jeweiligen Neurons j über alle Inputs x_i des vorherigen Layers, die individuelle Gewichtung ω_{ij} zwischen beiden und einem optionalen Bias b_i:

$$\begin{aligned} h_1 &= \omega_{11} \cdot x_1 + b_1 + \omega_{12} \cdot x_2 + b_2 + \dots \\ h_2 &= \omega_{21} \cdot x_1 + b_1 + \omega_{22} \cdot x_2 + b_2 + \dots \\ &\vdots \end{aligned} \tag{2.1}$$

Um den finalen Output y_i des i-ten Neurons zu berechnen, wird das Ergebnis der vorherigen Gleichung nun der Aktivierungsfunktion Φ übergeben, welche eine nicht-lineare Komponente darstellt:

$$y_1 = \Phi(h_1, \theta) \tag{2.2}$$

Je nach Aktivierungsfuntkion, können noch zusätzliche Parameter θ benötigt werden [10]. Es gibt zahlreiche mögliche Aktivierungsfunktionen, wie die Sigmoid-Funktion (welche in Abb. 2.1 verwendet wurde [9]), der Tangens Hyperbolikus oder die ReLU-Funktion, welche den Input bei positivem Vorzeichen unverändert lässt und sonst null zurückgibt.
Bei Convolutional Layers funktioniert die Berechnung etwas anders. Dort fährt ein sogenannter Kernel (eine Matrix, die kleiner als der Input ist) den zweidimensionalen Input Schritt für Schritt ab und berechnet, wie zuvor erwähnt, über Faltung Stück für Stück den neuen Output (sog. Feature Map) des Neurons. Verschiedene Kernels erzeugen verschiedene Feature Maps und können so z.B. die Konturen eines Bildes ermitteln [8].
In der zweiten Phase, der sogenannten Backward-Propagation, findet dann die Anpassung der Gewichtungen (und eventuell anderer Parameter) statt. Die Basis des eigentlichen Lernens bildet nun die sogenannte Cost- oder Loss-Funktion Ψ, welche ein Maß für die Fehlerrate von Vorhersagen des Netzwerkes darstellt. Bei (supervised) Regressionsproblemen kann der Loss z.B. über den gemittelten, quadrierten Fehler (MSE, Mean Square Error) berechnet werden [11].
Anschließend passt eine Optimierungsfunktion (bzw. ein Optimizer) nun die Gewichtungen unter den Neuronen so an, dass der Loss minimiert wird. Es gibt zahlreiche Optimizer, wie z.B. SGD (Stochastic Gradient Descent), welcher über den Gradienten der Lossfunktion die Gewichtungen verändert:

$$\omega_{neu} = \omega_{alt} - \eta \frac{d\Psi}{d\omega} \tag{2.3}$$

Eine wichtige Rolle spielt dabei die Lernrate bzw. *Learning Rate* η, welche die Schrittweite vorgibt, mit der man optimiert. Abbildung 2.4 zeigt den Verlauf der Cost- bzw. Lossfunktion bei verschiedenen Learning Rates. Wählt man η zu klein (die blauen Punkte), besteht die Gefahr, dass die Optimierung in einem lokalen Minimum endet, obwohl die Gewichtungen weiter reduziert werden könnten, oder das globale Minimum nicht erreicht wird, weil nicht genügend Epochen durchlaufen wurden. Bei einer zu großen Learning Rate (die grünen Punkte) wiederum könnte das globale Minimum überschritten und damit ebenfalls nicht nah genug erreicht werden.

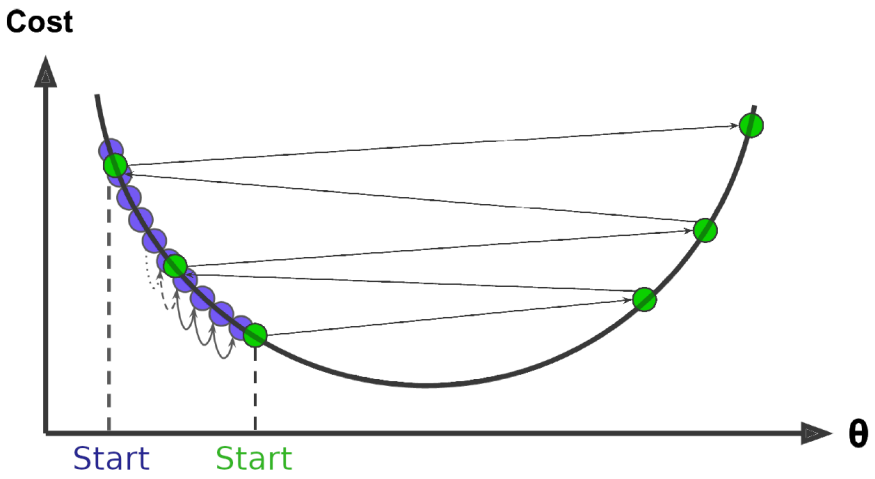


Abbildung 2.4: Die Losskurve mit verschiedenen Learning-Rates, blau: Die Lernrate ist zu klein, grün: Die Lernrate ist zu groß, Abbildungen aus [11]

Andere Optimizer, wie Adam, haben keine feste Learningrate für alle Parameter, sondern jeweils eigene, deren Wert auch an den entsprechenden Gradienten angepasst wird. Zusätzlich kann man mit einem sogenannten „Scheduler“ die Learning Rate auch im Laufe des Trainings variieren und durch eine immer kleiner werdende Schrittweite versuchen, sich dem globalen Minimum anzunähern [10].

2.3 Evaluation des Lernvorgangs

Wiederholt man beide Phasen über mehrere Epochen, optimiert sich das Netzwerk im besten Fall immer mehr. Die Gewichtungen werden also so angepasst, dass der Loss kleiner wird. Die Vorhersagen werden sowohl mit Input-Daten getroffen, die das Netzwerk bereits zum Lernen verwendet hat (Trainings-Daten) als auch mit bisher unbekannten (Validierungs-Daten), um zu testen, ob das Netz auch mit neuen Daten gute Ergebnisse erzielt. Verbessern sich die Prognosen im Laufe des Trainings nicht, da die Struktur des Netzes zur Lösung des jeweiligen Problems noch unzureichend ist, spricht man von Underfitting bzw. davon, dass das Netz divergiert (siehe die linke Grafik in Abb. 2.5). In den meisten Fällen kann es aber dazu kommen, dass die Vorhersagen der Trainingsdaten zwar mit jeder Epoche besser werden, doch die Vorhersagen der Validierungsdaten stagnieren oder sich gar verschlechtern - dann spricht man von Overfitting (siehe die rechte Grafik in Abb. 2.5). Der Grund hierfür liegt meist in der Komplexität des Netzes, welches sich zu sehr auf die Strukturen der bekannten Trainingsdaten verlässt [10].

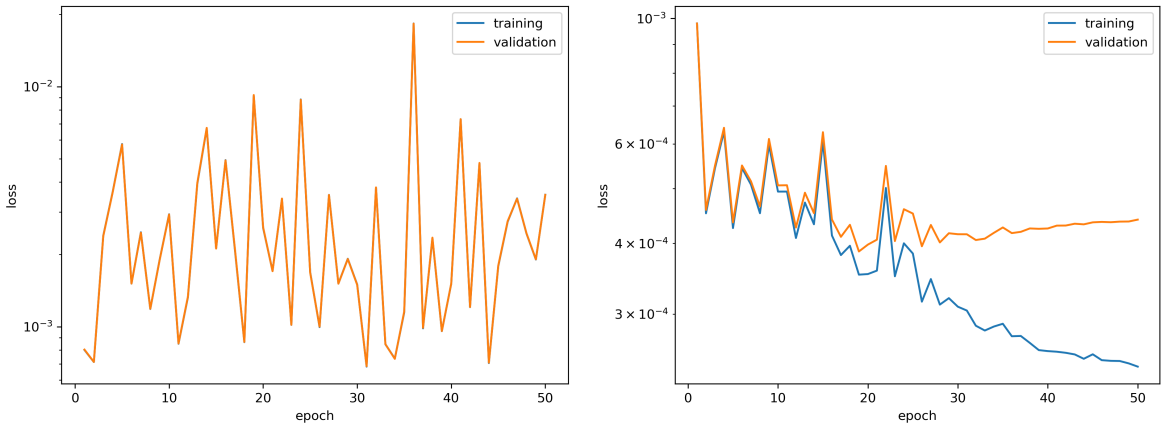


Abbildung 2.5: Links: Underfitting, Rechts: Overfitting (Eigene Grafik)

Ein weiteres, häufig auftretendes Problem ist das der verschwindenden Gradienten. Es kann vorkommen, dass die in der Back-Propagation berechneten Gradienten der Loss-Funktion mit jeder Epoche immer kleiner werden und dadurch keine ordentliche Optimierung mehr erfolgen kann. Aber auch das Gegenbeispiel ist möglich, wenn die Gradienten immer größer werden und das Netzwerk divergiert. Dann spricht man von explodierenden Gradienten [11]. Es gibt zahlreiche Strategien zur Optimierung des Netzes bzw. Vermeidung dieser Probleme. Eine Normierung der Inputs und die Wahl der richtigen Aktivierungsfunktion helfen, das Problem verschwindender Gradienten vorzubeugen. Ebenfalls hilfreich kann hier eine passende Initialisierung der Gewichtungen zu Beginn des Trainings sein. Zur Vermeidung von Overfitting kann das sogenannte "Drop-Out"(siehe [12]) helfen, bei der ein zufälliger Teil der Neuronen eines Layers deaktiviert werden, so dass sich das Netz nicht auf feste Strukturen verlassen kann und die Neuronen gezwungen sind immer wieder auf neue Verbindungen einzugehen. Eine andere Strategie zur Vermeidung von Overfitting ist nur die Modellparameter zu speichern bei denen der Loss auf dem Validierungsset ein neues Minimum erreicht und sonstige entstehende Veränderungen zu ignorieren. Trifft nach einer vorgegebenen Anzahl an Epochen keine Verbesserung mehr ein, kann der Lernprozess auch abgebrochen werden um Rechenleistung zu sparen. Dann spricht man vom sogenannten „Early Stopping“ [11].

3 Vorgehen

3.1 Ziel und Annahmen

Ziel dieser Arbeit ist die Rekonstruktion der Verteilung von Luminosität und Radius von Gammaquellen. Dabei wird untersucht, ob eine Verbesserung der Rekonstruktion erzielt werden kann, wenn neben Quellen bekannter Ausdehnung und Entfernung auch andere Quellen berücksichtigt werden, bei denen Ausdehnung und/oder Entfernung unbekannt sind, jedoch stochastisch abgeschätzt werden. Es wird angenommen, dass Luminosität und Radius voneinander unabhängig sind und einem Potenzgesetz folgen, so dass Gleichung 3.1 die kombinierte Verteilungsfunktion $P(L, R)$ beschreibt. [13]

$$P(L, R) = N \left(\frac{L}{L_0}\right)^{\alpha_L} \left(\frac{R}{R_0}\right)^{\alpha_R} \tag{3.1}$$

Hier sind $L_0 = 10^{34}\,\mathrm{ph/s}$ und $R_0 = 1\,\mathrm{pc}$ Skalenfaktoren, sowie N ein Normierungsfaktor, der von den festgelegten Grenzen für L und R abhängt. Damit hängt die konkrete Verteilungsfunktion in erster Linie von den Parametern α_L und α_R ab, die es zu ermitteln gilt. Wie bereits in Kapitel 1 angesprochen, beinhaltet der HGPS Katalog lediglich 16 Gammaquellen von denen Entfernung und Ausdehnung bekannt sind, so dass sich Radius und Luminosität berechnen ließen. Für eine verlässliche Statistik sind das zu Wenige. In [13] wurden künstliche Quellpopulationen simuliert, der $L \times R$ Phasenraum in $0,1 \times 0,1$ große Bereiche (auf einer logarithmischen Skala) eingeteilt und mittels einer Maximum-Likelihood-Methode unter Berücksichtigung des Beobachtungsbias die Parameter α_L und α_R abgeschätzt. In dieser Arbeit soll die Rekonstruktion der Parameter über Machine-Learning geschehen. Auch dafür werden, wie in [13], zuerst künstliche Gammaquellpopulationen erstellt und unter Berücksichtigung des Beobachtungsbias Datasets zu zufällig festgelegten Parametern konstruiert, die aus verschiedenen Histogrammen für je ein Parameterpaar bestehen. Anschließend wird ein CNN mit diesem Dataset darauf trainiert, die Zusammenhänge zwischen den Histogrammen und den richtigen Modellparametern α_L und α_R zu lernen. Im letzten Schritt kann dann aus den HGPS-Katalog Quellen ein Histogramm konstruiert werden. Auf dessen Grundlage trifft das CNN anschließend eine Vorhersage für die zu erwartenden Parameter und damit der zu ermittelnden Verteilungsfunktion.

3.2 Räumliche Verteilungsmodelle

Wie bereits erwähnt liegt im HGPS Katalog ein großer Beobachtungsbias vor (siehe auch die inhomogene Belichtungszeit in Abbildung 1.2), welcher unter anderem auch räumlich bedingt ist. Abbildung 3.1 zeigt die Verteilung der Quellen mit bekannter Entfernung (blaue Kreuze, das rote Kreuz stellt die Position der Sonne dar) und in grau sind dabei Spiralarme der Milchstraße angedeutet.

Um dem Problem des räumlichen Beobachtungsbias zu begegnen wurden in [13] mehrere

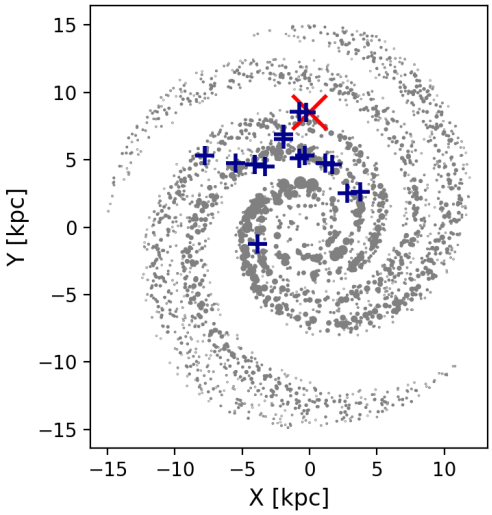


Abbildung 3.1: blau: Positionen der Gammaquellen des HGPS mit bekannter Entfernung, rot: Position der Sonne, grau: simulierte Positionen für Gammaquellen nach Modell mSp4, (Eigene Grafik)

Verteilungen modelliert, welchen sich im Zuge dieser Arbeit bedient wird. Da sich mehrheitlich Gammaquellen als Supernovaüberrest (SNR, SuperNova Remnant) oder als PulsarWindNebel (PWN) kategorisieren lassen, kann man auf Grundlage der Verteilung von SNRs bzw. PWNs zwei radialsymmetrische Modelle konstruieren, deren Quelledichte ρ sich wie folgt ergibt:

$$\rho(r,z) = \left(\frac{r + r_{off}}{R_\odot + r_{off}}\right)^{\alpha} \exp\left(-\beta \frac{r - R_\odot}{R_\odot + r_{off}}\right) \exp\left(-\frac{|z|}{z_0}\right) \tag{3.2}$$

Dabei steht r für den Abstand zum galaktischen Zentrum und z für die Höhe über der galaktischen Ebene. Man erhält die Modelle mSNR mit Parametern aus [14], und mPWN mit Parametern aus [15], deren Verteilung ungefähr aussieht wie (a) in Abbildung 3.2. Die Skalenhöhe z_0, der Formparameter α, der Ratenparameter β und r_{off}, welcher eine nicht-verschwindende Dichte im Zentrum berücksichtigt, sind vom jeweiligen Modell abhängig und in 3.1 aufgeführt. $R_\odot$ beschreibt den Abstand der Sonne zum galaktischen Zentrum.

Modell	$R_\odot$ [kpc]	r_{off} [kpc]	α	β	z_0 [kpc]
mSNR	8,5	0	1,09	3,87	0,085
mPWN	8,5	0,55	1,64	4,01	0,18

Tabelle 3.1: Parameter für die Modelle mSNR und mPWN

Im Gegensatz dazu ist auch denkbar, dass die Verteilung der Gammaquellen mit der Struktur der Milchstraße und der Verteilung interstellarer Materie zusammenhängt. Eine vierarmige Verteilung gibt das Modell mSP4 vor, welches mit Parametern aus [16] dieser, vom Azimut Φ abhängigen, Gleichung folgt:

$$\rho(r,\Phi,z) = \sum_{i=1}^{4} A_i \exp\left(-\frac{1}{\delta^2}\left(\Phi - \frac{\ln\left(\frac{r}{a_i}\right)}{\beta_i}\right)^2\right) \exp\left(-\frac{|r - R|}{\sigma_r}\right) \exp\left(-\frac{z^2}{2\sigma_{z,2}^2}\right) \tag{3.3}$$

In Abbildung 3.2 ist dieses Modell in (c) dargestellt. Die radiale Abhängigkeit wird durch eine Skalenlänge σ_r und ein lokales Maximum bei R beschrieben. Die azimutale Abhängigkeit wiederum wird durch die Skalenlänge δ, sowie dem Steigungswinkel β_i und der Orientierung α_i des jeweiligen Spiralarms beschrieben. Die Skalenhöhe in z-Richtung definiert $\sigma_{z,2}$; A_i stellt jeweils einen Normierungsfaktor dar. Diese Parameter unterscheiden sich für die verschiedenen Arme, wie Tabelle 3.2 zeigt.

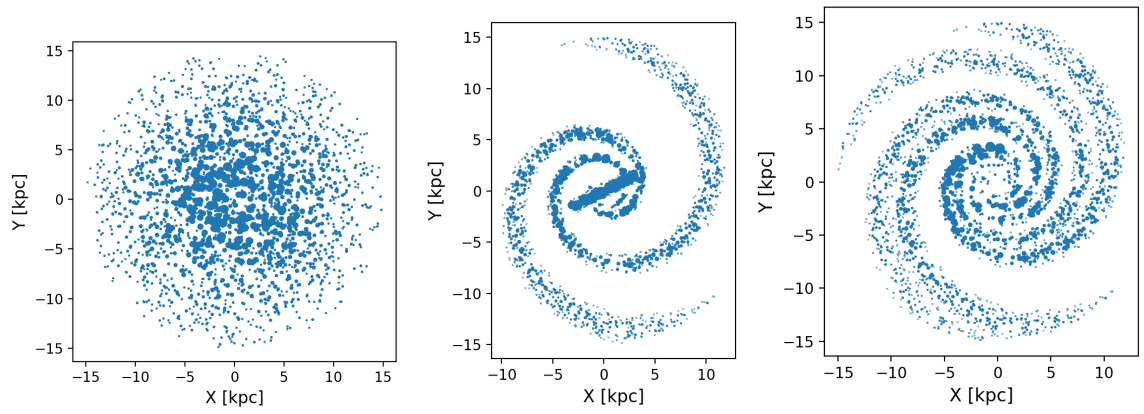
Spiralarm	β_i	α_i	R [kpc]	σ_r [kpc]		$\sigma_{z,2}$ [kpc]	δ [deg]	A_i
				$r < R$	$r > R$			
Sagittarius-Carina	0,242	0,246	2,9	0,7	3,1	0,07	15	169
Scutum-Crux	0,279	0,608	2,9	0,7	3,1	0,07	15	266
Perseus	0,249	0,449	2,9	0,7	3,1	0,07	15	339
Norma-Cygnus	0,240	0,378	2,9	0,7	3,1	0,07	15	176

Tabelle 3.2: Parameter für die Spiralarme in den Modellen mSp4 und mSp2B

Auch denkbar ist, dass die Verteilung der Gammaquellen der Struktur des galaktische Balkens der Milchstraße mit einer Ausdehnung von $l_b = 3.5\,\text{pc}$ folgt (siehe [17]). Das Modell mSp2B verfolgt genau diesen Ansatz mit der Verteilung aus 3.3, allerdings nur für 2 Arme (den Scutum-Crux und Perseus Arm) und der Balkenquellendichte ρ_{bar}, die durch folgenden Zusammenhang beschrieben wird:

$$\rho_{bar}(r, \Phi, z) = \begin{cases} A_{bar} \exp\left(-\frac{z^2 + r^2(\sin\Phi - \cos\Phi \sin\theta)^2}{\sigma_{z,1}^2}\right) & \text{für } r < l_b \\ 0 & \text{sonst} \end{cases} \tag{3.4}$$

In Abbildung 3.2 ist dieses Modell in (b) dargestellt. Dabei sind $\theta = 30°$ der Winkel zur Linie zwischen der Sonne und dem galaktischem Zentrum, $\sigma_{z,1} = 0.31\,\text{pc}$ die z-Skalenhöhe und $A_{bar} = 364$ ein Normierungsfaktor, der dafür sorgt, dass Balken und Spiralarme jeweils gleich zur Verteilung beitragen.



(a) (b) (c)

Abbildung 3.2: Räumliche Verteilungsmodelle: (a) mSNR/mPWN, (b) mSp2B, (c) mSp4, (Eigene Grafik)

Zum Optimieren des CNNs wurde hauptsächlich das Modell mPWN verwendet; getestet und rekonstruiert wurde dann auch mit den übrigen Modellen.

3.3 Verwendete Tools

In der Arbeit wurde mit Python (Version 3.8.5) und Anaconda (Version 2020.11) gearbeitet. Folgende Packages wurden dabei verwendet:

- h5py (Version 2.10.0)
- numpy (Version 1.19.2)
- pytorch (Version 1.7.0)
- scipy (Version 1.5.2)
- mkl_random (Version 1.1.1)
- pandas (Version 1.1.3)
- matplotlib (Version 3.3.2)

3.4 Erstellen des Datasets

In dieser Arbeit wurden zwei Ansätze verfolgt. Zum einen werden nur detektierbare Quellen mit bekannter Ausdehnung und Entfernung berücksichtigt (im Folgenden Methode 1 genannt), wie auch schon in [13] nur solche verwendet wurden, zum anderen werden alle detektierbaren Quellen berücksichtigt. Eventuelle fehlende Größen (unbekannte Entfernung oder nicht messbare Ausdehnung) werden hier entsprechend stochastisch abgeschätzt (Methode 2), um zu testen ob eine solche Abschätzung sinnvoll ist um zusätzliche Informationen aus den simulierten Populationen zu extrahieren und verlässlichere Vorhersagen zu generieren. Zuerst werden allerdings in beiden Fällen im gesamten Bereich der Milchstraße, angenähert als ein galaktischer Zylinder mit Radius 15 kpc und Höhe 2 kpc, zufällige Quellpositionen mit einer mittleren freien Weglänge von 50 pc festgelegt. Je nach verwendetem räumlichen Verteilungsmodell wurden die Positionen dann unterschiedlich gewichtet. Für eine Quellpopulation mit zufällig festgelegten Parametern α_L und α_R müssen dann einer bestimmten Anzahl an Positionen enstprechende Luminositäten und Radien zu geordnet werden. Dabei ist die Gesamtanzahl an Quellen innerhalb der Population so zu bestimmen, dass letztendlich unter Berücksichtigung des Beobachtungsbias, ungefähr die gleiche Anzahl an Quellen wie im HGPS Katalog detektiert werden. Für Methode 1 entspricht das den 16 Quellen bekannter Entfernung und Ausdehnung und für Methode 2 sind das 68 detektierbare Quellen. Um die Gesamtzahl abschätzen zu können, wird dafür zunächst auf einem zweidimensionalen Grid mit verschiedenen Werten der Parameter α_L und α_R benötigte Populationsgrößen berechnet. Detektierbar sind alle Quellen, deren Ausdehnung 1° nicht überschreitet und deren auf der Erde eintreffender Strahlungsfluss über der jeweiligen Sensitivität des HGPS liegt, sowie deren Positionen überhaupt im Gesichtsfeld des HGPS liegen. Für Methode 1 wird auch hier entsprechend eine Vorauswahl von Quellen getroffen, deren Ausdehnung messbar ist (also über 0.08° liegt) und deren Entfernung bekannt ist. Dafür wird angenommen, dass wie im HGPS von einem Drittel der Quellen die Entfernung bekannt ist. Welche Quellen

genau als Quellen bekannter Entfernung behandelt werden, wird zufällig ausgewählt. Das somit erhaltene Gitter wird genutzt, um Populationsgrößen für Parameter, die zwischen den vorgegebenen Gitterpunkten liegen, zu interpolieren. Damit kann nun eine Quellpopulation für ganz zufällige Parameter in einem Bereich von $-3 < \alpha_L < 0$ und $-3 < \alpha_R < 0$ für Methode 1 bzw. einem Bereich von $-3 < \alpha_L < 2$ und $-3 < \alpha_R < 2$ für Methode 2 erstellt werden. Der Bereich wurde für Methode 2 erweitert, da die Rekonstruktionen des Radiusparameter hier positiv wurden, worauf in Kapitel 5 näher eingegangen wird. Die Grenzen für Luminosität und Radius werden denen der Quellen bekannter Entfernung und Ausdehnung im HGPS angepasst. Das bedeutet, die Luminositäten gehen von 10^{32} ph/s bis 10^{35} ph/s, die Radien von $10^{0.4}$ pc bis $10^{1.7}$ pc.

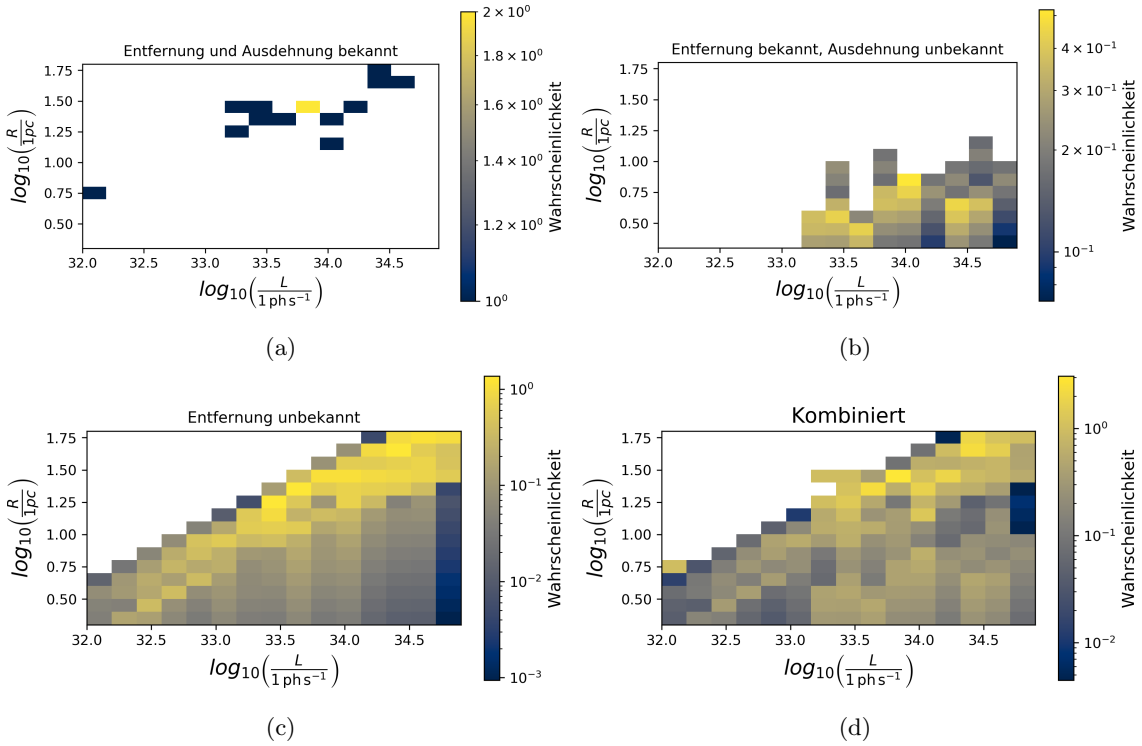


Abbildung 3.3: Verschiedene L-R-Histogramme: (a) Quellen mit bekannter Entfernung und Ausdehnung (Methode 1), (b) Quellen mit bekannter Entfernung, aber unbekannter Ausdehnung, (c) Quellen mit unbekannter Entfernung, (d) alle Histogramme kombiniert (Methode 2), (Eigene Grafik)

Aus dieser Population wird nun das Histogramm als Teil des Datasets für das CNN berechnet. Auf den oben beschriebenen Luminositäts- und Radiusbereichen werden jeweils 15 logarithmische Unterteilungen als Bins für das Histogramm festgelegt. Es ergeben sich nun drei Möglichkeiten für eine Quelle.

Bekannte Entfernung und Ausdehnung
In diesem Fall, welcher für Methode 1 ausschließlich vorkommt, sind Luminosität L und Radius R einfach über geometrische Relationen (siehe Gleichungen 3.5 und 3.6) zu berechnen. Die Quellen bilden je einen Punkt im Histogramm, wie in Abbildung 3.3 bei (a) dargestellt

ist, wobei D die Entfernung, E die Ausdehnung und F der Strahlungsfluss ist.

$$R = 2D \sin(E) \tag{3.5}$$

$$L = 4\pi D^2 F \tag{3.6}$$

Entfernung bekannt, Ausdehnung unbekannt
Dieser Fall betrifft nur Methode 2. Hier lässt sich die Luminosität noch über Gleichung 3.6 berechnen, doch der Radius kann nur abgeschätzt werden. Mit der Auflösung 0.08°, aus der nach 3.5 eine Obergrenze für den Radius resultiert, kann eine stetige Gleichverteilungsfunktion für die Radien bestimmt werden, welche sich als Säulen im Histogramm äußert (siehe (b) in Abbildung 3.3).

Entfernung unbekannt
Auch dieser Fall ist nur für Methode 2 relevant. Ist die Entfernung aber unbekannt, wie bei den meisten Quellen im HGPS-Katalog, so müssen Luminosität und Radius entlang der Sichtlinie der Quelle abgeschätzt werden. Dafür werden für verschiedene Entfernungen, welche je nach Verteilungsmodell unterschiedlich gewichtet werden, Luminosität und Radius nach den obigen Gleichungen berechnet. Somit bildet jede Quelle, repräsentiert als Verteilungsfunktion, eine diagonale Linie im Histogramm ab. Zusammen ergibt sich dann z.B. ein Histogramm wie (c) in Abbildung 3.3.

Vereint man nun alle Histogramme, ergibt sich für Methode 2 das kombinierte Histogramm (d). Dieses bildet zusammen mit dem zugehörigen Parametern α_L und α_R einen Eintrag im Dataset. Für Methode 1 sehen die Histogramme wie (a) in Abbildung 3.3 aus. Damit das CNN später zuverlässige Vorhersagen macht, müssen genügend Daten vorliegen, so dass ein Dataset ungefähr 100.000 Histogramme für Methode 1 und ungefähr 160.000 Histogramme für Methode 2 beinhalten sollte, von denen ein Drittel als Validierungsdataset und der Rest als Trainingsdataset verwendet wird. Das Validierungsset wird anschließend auch für die Evaluierung der finalen Performance des Netzes verwendet.

4 Aufbau und Optimieren des CNN

4.1 Struktur der Netzes

Da die Aufgabe des Netzes ist aus zweidimensionalen Inputs (Histogrammen) einen eindimensionalen Vektor mit den Parametern α_L und α_R zu rekonstruieren, wird ein CNN benötigt, dessen letzter Layer ein fc-Layer ist. Die Anzahl und Struktur der convolutional werden im Folgenden angepasst.
Das Netz durchläuft im Training 50 Epochen. Dabei gibt es pro Epoche zwei Phasen: Die Trainingsphase, in der die Forward- und Backward-Propagation und damit die Optimierung des Netzes stattfinden, und eine Evaluationsphase, in der nur die Forward-Propagation durchlaufen wird und die Vorhersagekraft einmal anhand der Trainings- und einmal anhand der Validierungsdaten getestet wird. Der Loss wird dafür über den MSE (Mean Square Error, mittlerer quadrierter Fehler) berechnet und pro Epoche gespeichert. Um eventuellem Overfitting zu begegnen, werden während der 50 Epochen nur die Netzparameter gespeichert, bei denen der Validierungs-Loss ein neues Minimum erreicht hat. Sollten sich die Performance anschließend also verschlechtern, bleiben nur die bisher optimalen Parameter erhalten.
Ein Scheduler passt dabei die Learningrate im Verlauf automatisch an, was wiederum den Lernprozess verbessert, da man so das Problem zu großer oder zu kleiner Lernraten in den Griff bekommen kann. Der verwendete Scheduler "ReduceLROnPlateu"reduziert die Lernrate um den Faktor $0,5$, wenn innerhalb von 5 Epochen keine signifikante Verbesserung eintrat. Außerdem wird die Drop-Out Strategie im fc-Layer angewandt, so dass 50% der Inputs gleich null gesetzt werden. Damit soll zusätzlich Overfitting vermieden werden.

4.2 Normierung des Datasets

Sind die Inputs des Netzwerks und der einzelnen Layer nicht normiert, kann das die Leistung eines neuralen Netzes verschlechtern, da durch eine Normierung das Problem verschwindender bzw. explodierender Gradienten vermindert werden kann [11]. Zur Normierung der Inputs eines jeden Layers werden die Daten einer sogenannten Batch-Normalization, wie sie in [18] vorgestellt wurde, unterzogen. Dabei werden nach Gleichung 4.1 der Mittelwert μ_B und die Standardabweichung σ_B der unnormierten Inputs x_i jeweils eines Mini-Batches berechnet. Ein Mini-Batch ist ein Paket an Input-Daten der Größe m_B. Die neuen Inputs $x_{i,norm}$ werden normalisiert, mit Mittelwert$= 0$ und Standardabweichung$= 1$; $\epsilon = 10^{-5}$ zur Vermeidung von Division durch null. Anschließend werden die normalisierten Inputs elementweise mit einem Parameter γ multipliziert und mit β addiert um den Output z_i zu erhalten. Die optimalen Werte dieser Parameter (γ und β) werden in der Back-Propagation gelernt. Während des Trainings werden bereits der finale Mittelwert μ und die finale Standardabweichung σ als exponentiell gleitende Mittelwerte abgeschätzt, damit diese dann in der Evaluationsphase die von den jeweiligen Mini-Batches abhängigen Mittelwerte und Standardabweichungen ersetzen können.

$$\begin{aligned} \mu_B &= \frac{1}{m_B}\sum_{i=1}^{m_B} x_i \\ \sigma_B &= \frac{1}{m_B}\sum_{i=1}^{m_B}(x_I - \mu_B)^2 \\ x_{i,norm} &= \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\ z_i &= \gamma \otimes x_{i,norm} + \beta \end{aligned} \tag{4.1}$$

Natürlich kann auch das Dataset an sich normiert werden. Besonders wenn keine Batch-Normalization verwendet wird, kann dies die Vorhersagekraft des Netzes verbessern. Abbildung 4.1 zeigt den quadrierten Fehler der Vorhersagen für verschiedene Labels (Vorgabe-Parameter). Gut zu erkennen ist, dass das Netz mit Batch-Normalization (die blaue Kurve) bessere Vorhersagen (Vorhersagen mit einem geringeren Fehler) trifft, als die Netze ohne.

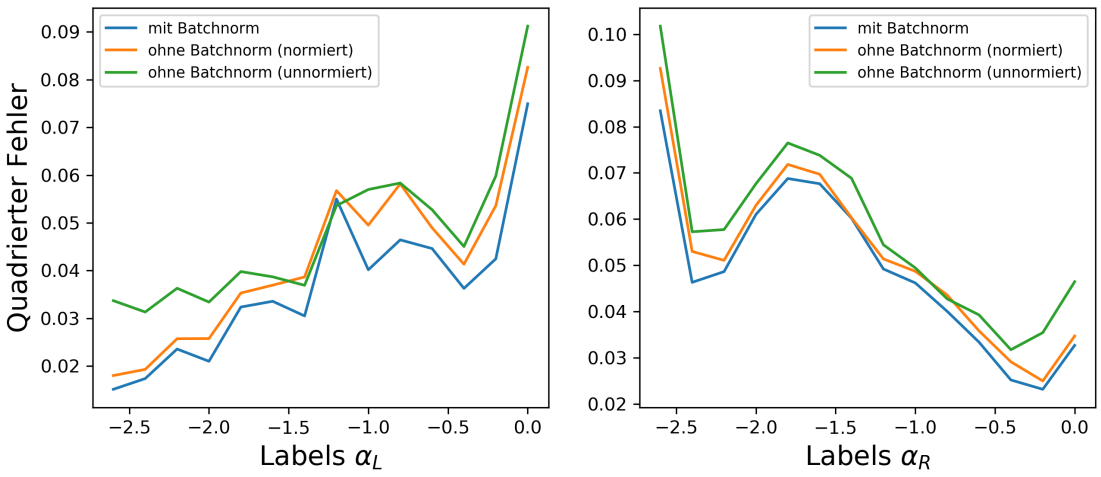


Abbildung 4.1: Einfluss der Batch-Normalization auf den quadrierten Fehler der Vorhersagen

4.3 Initialisierung der Gewichte

Es gibt zahlreiche Möglichkeiten, die Gewichtungen vor dem ersten Training zu initialisieren, welche jede für sich die Vorhersagekraft des Netzes beeinflussen. Möglich ist es allen Gewichtungen den gleichen Wert geben, z.B. eins. Eine andere Möglichkeit ist die Gewichtungen zufällig zu verteilen, z.B. über eine stetige Gleichverteilung (Uniform-Distribution, standardmäßig auf einem Intervall von null bis eins) oder über eine Normalverteilung (standardmäßig mit einem Erwartungswert von null und einer Standardabweichung von eins). Neben diesen wurden auch zwei weitere Initialisierungen getestet. Die Glorot- oder Xavier-Initialisierung, vorgeschlagen in [19], und die He- oder Kaiming-Initialisierung, vorgeschlagen in [20]. Beide verfolgen das Ziel, das Problem der verschwindenden bzw. explodierenden Gradienten zu lösen, indem die Streuung der Outputs eines Layers der Streuung seiner Inputs

entsprechen soll. Dabei beschreibt fan_{in} die Anzahl an Inputs und fan_{out} die Anzahl an Neuronen bzw. Outputs. Beide Methoden sind jeweils anwendbar auf eine stetigen Gleichverteilung, mit Intervallgrenzen von $-a$ bis a siehe Gleichung 4.2, oder auf eine Normalverteilung, mit Erwartungswert null und Standardabweichung siehe Gleichung 4.3.

$$a_{Glorot} = \sqrt{\frac{6}{fan_{in} + fan_{out}}} \qquad a_{He} = \sqrt{\frac{3}{fan_{in}}} \tag{4.2}$$

$$\sigma^2_{Glorot} = \frac{2}{fan_{in} + fan_{out}} \qquad \sigma^2_{He} = \frac{1}{fan_{in}} \tag{4.3}$$

Betrachtet man den Verlauf des quadrierten Fehlers der Vorhersagen in Abbildung 4.2, sowie den gemittelten quadrierten Fehler (MSE) in Tabelle 4.1, ist zu erkennen, dass eine einfache Gleichverteilung der Gewichtungen und das Festlegen aller Gewichtungen auf eins, eher schlecht abschneiden. Die übrigen Initialisierungen lassen das Netz bessere Vorhersagen treffen, wobei sich die Methode von Glorot angewendet auf eine Normalverteilung als beste Strategie zur Initialisierung der Gewichtungen entpuppt.

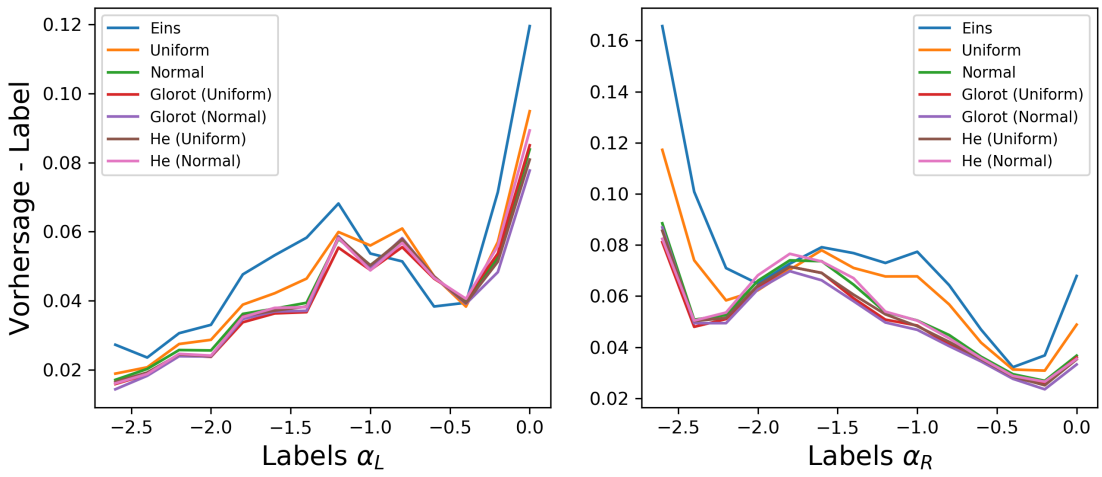


Abbildung 4.2: Quadrierter Fehler für verschiedene Initialisierungen, Fett markiert ist die Initialisierung mit dem kleinsten MSE

	Eins	Standard		**Glorot-Init**		He-Init	
		Uniform	Normal	Uniform	**Normal**	Uniform	Normal
MSE von α_L	0,0483	0,0433	0,0402	0,0390	**0,0387**	0,0395	0,0400
MSE von α_R	0,0766	0,0650	0,0554	0,0525	**0,0520**	0,0534	0,0552

Tabelle 4.1: Quadrierter Fehler (MSE) beider Parameter für die verschiedenen Initialisierungen

4.4 Optimieren der Layer- und Neuronenzahl

Generell gilt, je mehr Hidden Layer verwendet werden, desto mehr Eigenschaften können extrahiert werden und desto besser werden die Vorhersagen des Netzes. Allerdings erhöht sich mit jedem zusätzlichen Layern auch der Rechenaufwand und die Gefahr von Overfitting steigt. Daher gilt es also unter Abwägung dieser Faktoren eine gute Layer- und Neuronenzahl zu finden, welche nicht zuletzt vor allem von der Funktion des Netzes abhängen.

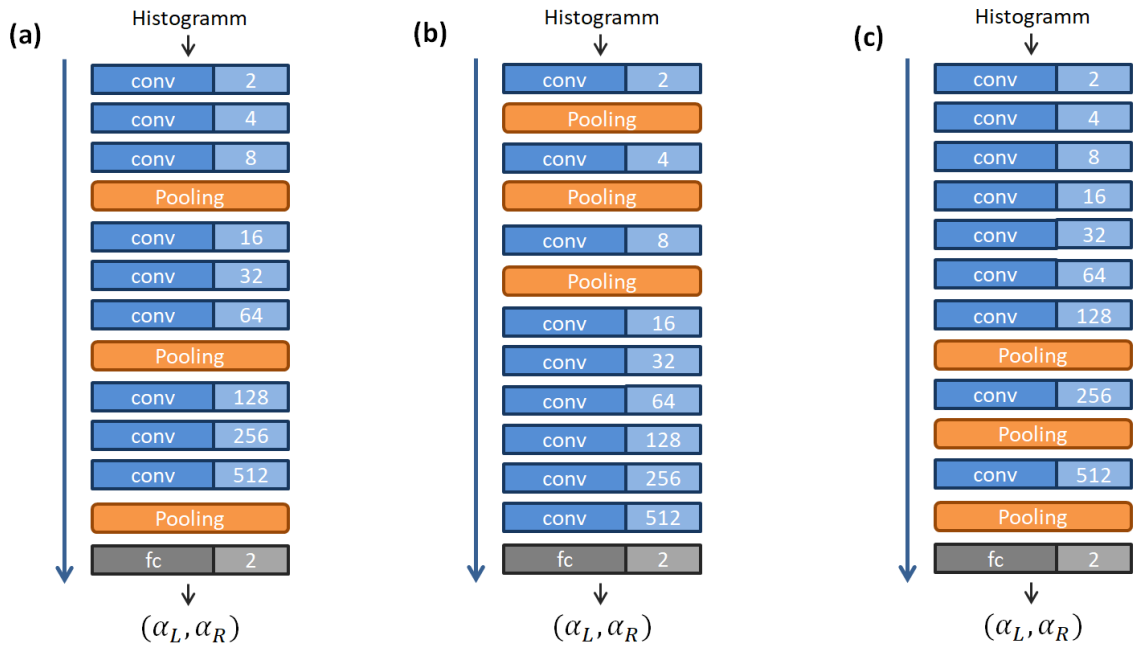


Abbildung 4.3: Die Architektur des CNN mit Pooling Layern an verschiedenen Stellen, (a) gleichverteilt, (b) am Anfang, (c) am Ende

Wie bereits in Kapitel 2 angesprochen sind Pooling-Layer unablässliche Bestandteile zur Reduzierung der Auflösung der Daten und damit auch des Rechenaufwandes. Da sich die Histogramme aus 15x15 Bins zusammensetzen, lässt sich ein Pooling mit einem 3x3 Kernel und einer Schrittweite von 2 Schritten insgesamt dreimal durchführen, da die Histogramme dabei auf Größen von 7x7, 3x3 und zuletzt 1x1 Bin reduzieren. Die nächste Frage, die sich damit stellt, ist das Finden der optimalen Position der Pooling-Layer. Je später das Pooling zum Einsatz kommen, desto mehr Informationen stehen den Hidden Layern am Anfang zur Verfügung. Aber desto höher wird auch der Rechenaufwand, wenn man von einer mit jedem Layer steigenden Neuronenzahl ausgeht. Daher empfiehlt [11] das Pooling gleichmäßig zu verteilen, welches auch in diesem Fall die beste Strategie ist, wie der Verlauf des quadrierten Fehlers der Vorhersagen (zu sehen in Abbildung 4.4) zeigt. Dort erfolgt bei einem Netzwerk mit 9 Hidden Layern (deren Neuronzahl sich jeweils verdoppelte) das Pooling einmal am Anfang jeweils nach einem Hidden Layer (siehe b in Abbildung 4.3), gleichverteilt aller 3 Hidden Layer (siehe a in Abbildung 4.3) und am Ende erst nach 7 Hidden Layern (siehe c in Abbildung 4.3). Für den letzteren Fall trat, wie zu erwarten, Overfitting auf, was in Abbildung 4.5 zu sehen ist.

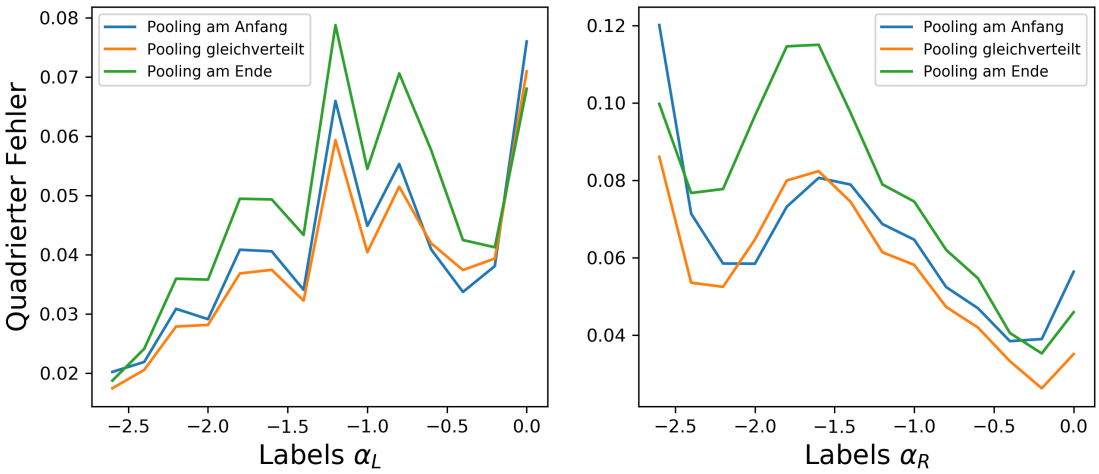
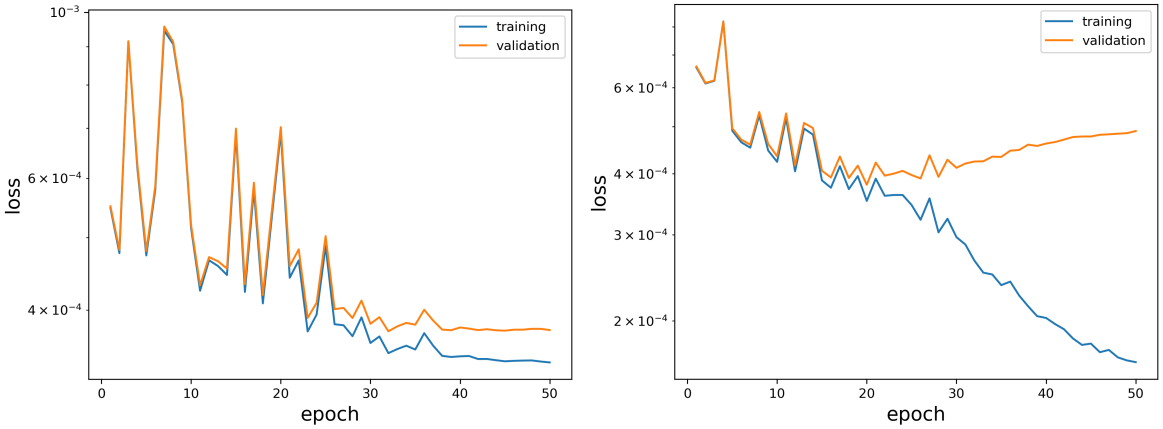


Abbildung 4.4: Quadrierter Fehler bei verschiedenen Positonen der Pooling Layer



(a) (b)

Abbildung 4.5: Loss für verschiedene Positionen der Pooling Layer bei 9 Hidden Layern, (a) für ein Pooling Layer aller 3 Hidden Layer, (b) für alle Pooling Layer erst am Ende nach 7 Hidden Layern

Für die Hierarchie der Layer, also wie sich die Neuronenzahl mit jedem Layer verändert, gibt es verschiedene Strategien, die für sich stark von der vorliegenden Problemstellung abhängen. Zum Beispiel empfiehlt [11] für allgemeine Bilder verarbeitenden Netzwerke die Neuronenzahl mit jedem Layer zu verringern, damit am Anfang viele niedrigschwellige Merkmale extrahiert werden, die am Ende in wenigen „high-level features“ aufgehen. Alternativ soll auch eine gleichbleibende Neuronenzahl ähnlich gute Ergebnisse erbringen. Neben diesen Ansätzen, wurde auch eine steigende Neuronenzahl mit jedem Layer probiert, bei der aus anfangs weniger niedrischwelligen Merkmalen später immer mehr komplexere extrahiert werden. Dies stellte sich in diesem Fall als beste Strategie heraus. Abbildung 4.6 zeigt in einem Netz mit 3 Hidden Layern die Differenz aus vorhergesagtem und Label- bzw. vorgegebenen Parameter für die drei besprochenen Strategien. Schwarz gepunktet ist die optimale Nulllinie. Da das Netz nur aus 3 Layern besteht, sind die Vorhersagen natürlich noch nicht optimal, aber es lässt sich bereits erkennen, dass eine mit jedem Layer aufsteigende Neuronenzahl im Vergleich besser abschneidet.

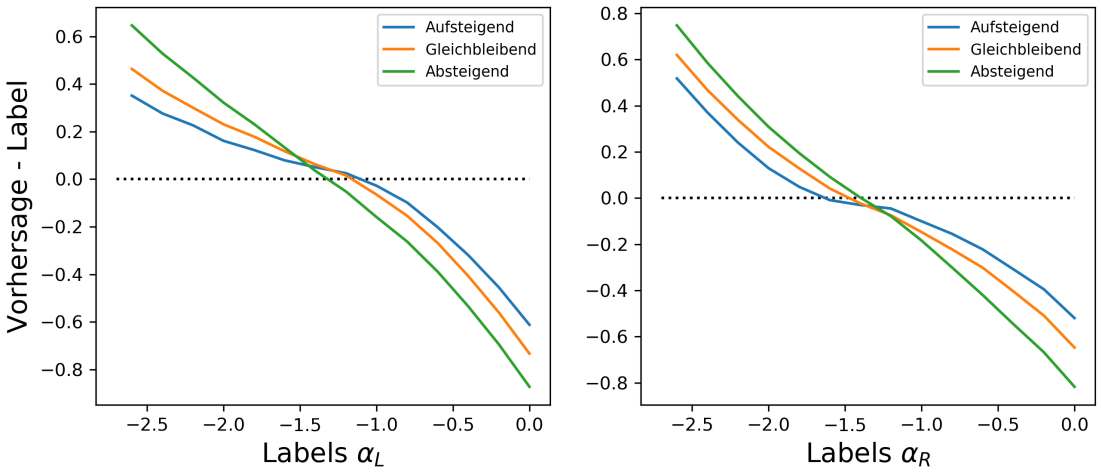


Abbildung 4.6: Verlauf der Vorhersagefehler bei verschiedenen Layer-Hierarchien

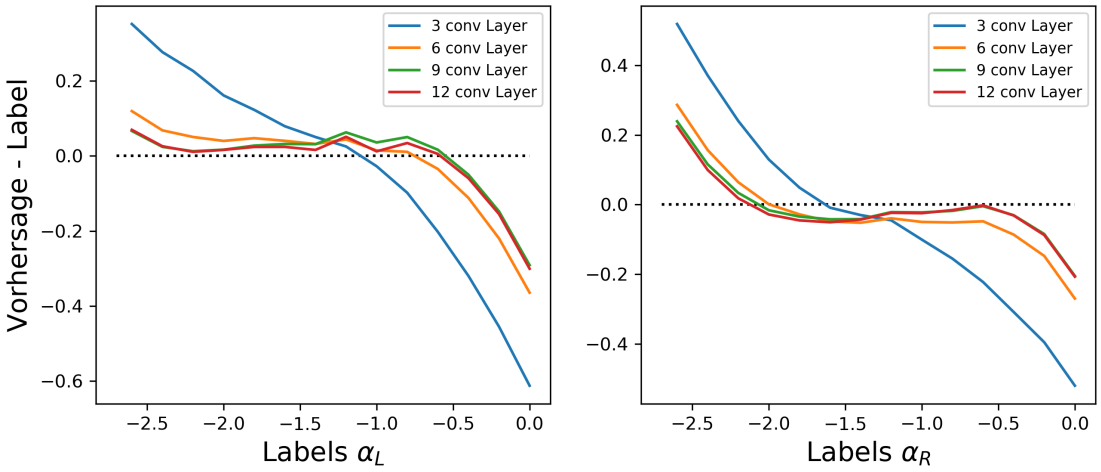


Abbildung 4.7: Fehlerverlauf bei verschiedenen conv. Layer-Anzahlen

Damit die Vorhersagen nun insgesamt besser werden, bleibt noch die Suche nach der optimalen Layer-Anzahl. Hier wurden Netze mit 3, 6, 9 und 12 Layern getestet; jeweils mit drei gleichverteilten Average-Pooling-Layern. Abbildung 4.7 zeigt die Entwicklung der Fehler (Differenz zwischen Vorhersage und Label) mit zunehmender Layer-Anzahl. Gut zu erkennen ist, dass die Vorhersagen mit zusätzlichen Layern besser werden, doch bereits zwischen 9 und 12 Layern kein großer Unterschied mehr besteht, in Bezug auf die Vorhersagekraft. Die Rechenzeit für 50 Epochen dagegen vervierfachte sich von 5 auf 20 Minuten.

4.5 Performance verschiedener Aktivierungsfunktionen

Als nichtlineare Komponente stehen viele mögliche Aktivierungsfunktionen zur Verfügung. Eine der aufgrund ihrer Einfachheit beliebtesten Funktionen ist die Rectified Linear Unit

(ReLU). Diese ist für negative Inputs null und gibt sonst den Input unverändert weiter. Da allerdings hier schnell das Problem der sterbenden Neuronen auftritt, d.h. der Output vieler Neuronen wird null, stehen alternativ auch die LeakyReLU bzw. die Parametric ReLU (PReLU) zur Verfügung. Hier wird bei negativem Vorzeichen der Input mit einem kleinen Parameter multipliziert, welcher bei LeakyRELU standardmäßig $\alpha = 0.1$ ist, und bei PReLU in der Backpropagation gelernt wird. Auch möglich sind der Tangens Hyperbolikus und die Sigmoid-Funktion. Hierbei tritt allerdings häufig das Problem verschwindender Gradienten auf, da die Ableitung dieser Funktionen, sowohl im positiv, aber auch im negativ Unendlichen null wird. Die Sättigung im positiv Unendlichen umgeht man mit den oben genannten Funktionen, aber auch mit der Exponential Linear Unit (ELU) vorgeschlagen in [21], und ihrer Weiterentwicklung der Self-normalizing Exponential Linear Unit (SELU) (siehe [22]). Hier werden negative Inputs durch eine Exponentialfunktion reguliert, wobei SELU eine skalierte Version von ELU darstellt, welche die Outputs des Netzwerks automatisch auf einen Mittelwert von null und einer Standardabweichung von eins normalisiert. Abbildung 4.8 zeigt alle besprochenen Aktivierungsfunktionen.

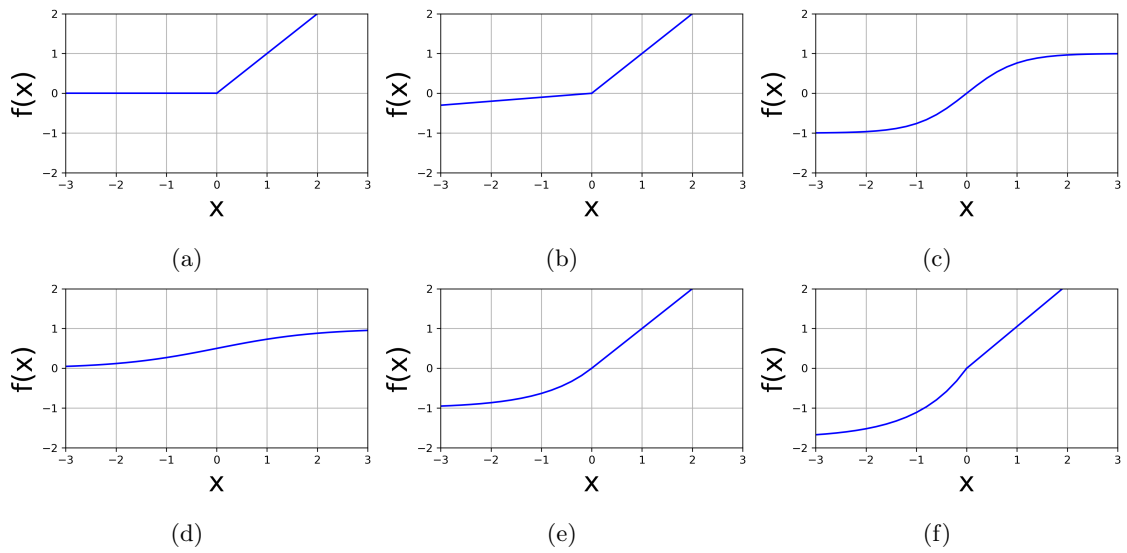


Abbildung 4.8: Verschiedene Aktivierungsfunktionen: (a) ReLU, (b) LeakyReLU bzw. PReLU, (c) Tangens-Hyperbolikus, (d) Sigmoid-Funktion, (e) ELU, (f) SELU, (Eigene Grafik)

Abbildung 4.9 zeigt den Verlauf der quadrierten Fehler der Vorhersagen für die besprochenen Funktionen. Es sind keine signifikanten Unterschiede zwischen den Graphen zu erkennen. Auch die gemittelten quadrierten Fehler, die in Tabelle 4.2 zu sehen sind, unterscheiden sich nur minimal, was keine klare Präferenz erkennen lässt.

	ReLU	LeakyReLU	PReLU	Tanh	Sigmoid	ELU	**SELU**
MSE von α_L	0.0391	0,0387	0,0388	0,0389	0,0386	0,0393	**0,0386**
MSE von α_R	0,0520	0,0517	0,0523	0,0514	0,0504	0,0507	**0,0507**

Tabelle 4.2: Quadrierter Fehler (MSE) beider Parameter für die verschiedenen Aktivierungsfunktionen, Fett markiert ist die mit der weiter gearbeitet wurde

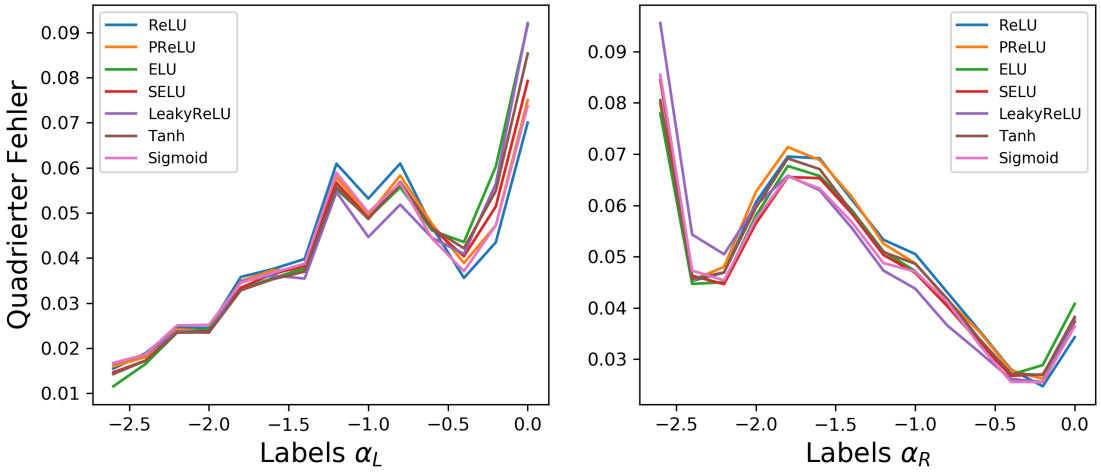


Abbildung 4.9: Verlauf der quadrierten Fehler bei verschiedenen Aktivierungsfunktionen

4.6 Einfluss verschiedener Optimizer

Neben dem schon in Kapitel 2 besprochenden Stochastic Gradient Descent (SGD), werden zwei weitere Optimizer getestet. Zum einen AdamW (aus [23]), welcher eine Weiterentwicklung von Adam (vorgeschlagen in [24]) darstellt, und zum anderen Rprop. Adam verwendet adaptive Lernraten und Momenta und basiert auf der Berechnung mittlerer und quadrierter Gradienten, was ihn auch mit großen Datenmengen zeit- und speichereffizient umgehen lässt. AdamW verwendet zusätzlich das sogenannte weight decay. Darunter versteht man, dass nun die neue Gewichtung nicht nur mit einem bestimmten Betrag subtrahiert wird, sondern auch ein wenig „verfällt“, d.h. auch wenn der Gradient null ist, kleiner wird. Die Idee dahinter ist, dass kleine Gewichtungen generell Overfitting zu vermeiden scheinen [23].
Der dritte Optimizer ist Resilient Propagation (Rprop, aus [25]). Dieser verwendet ein iteratives Verfahren in dem zur Bestimmung der aktuellen Anpassung der Gewichte die letzte Gewichtsänderung mit einbezogen wird. Auf die Erklärung der genauen Funktionsweise des Optimizers wird an dieser Stelle aber verzichtet, um nicht den Rahmen der Arbeit zu sprengen. Alle Optimizer werden jeweils mit drei verschiedene Lernraten (0,01 / 0,005 / 0,001) getestet. Abbildung 4.10 zeigt, wegen der Übersichtlichkeit nur für zwei Lernraten, den Verlauf der quadrierten Fehler für diese Optimizer. Man erkennt, dass SGD die größten Fehler in den Vorhersagen hat, wohingegen AdamW am besten abschneidet. Das zeigen auch die gemittelten quadrierten Fehler (MSE) für alle Lernraten in Tabelle 4.3. Es scheint, dass AdamW bei eher hohen Lernraten von 0,005 oder 0,01 den besten Optimizer für das CNN bildet.

Lernrate	SGD		Rprop		**AdamW**	
	MSE α_L	MSE α_R	MSE α_L	MSE α_R	MSE α_L	MSE α_R
0,01	0,0413	0,0573	0,0469	0,0690	**0,0382**	**0,0507**
0,005	0,0420	0,0610	0,0446	0,0677	0,0387	0,0510
0,001	0,0444	0,0662	0,0419	0,0654	0,0397	0,0544

Tabelle 4.3: Mean Squared Error beider Parameter für die verschiedenen Optimizer

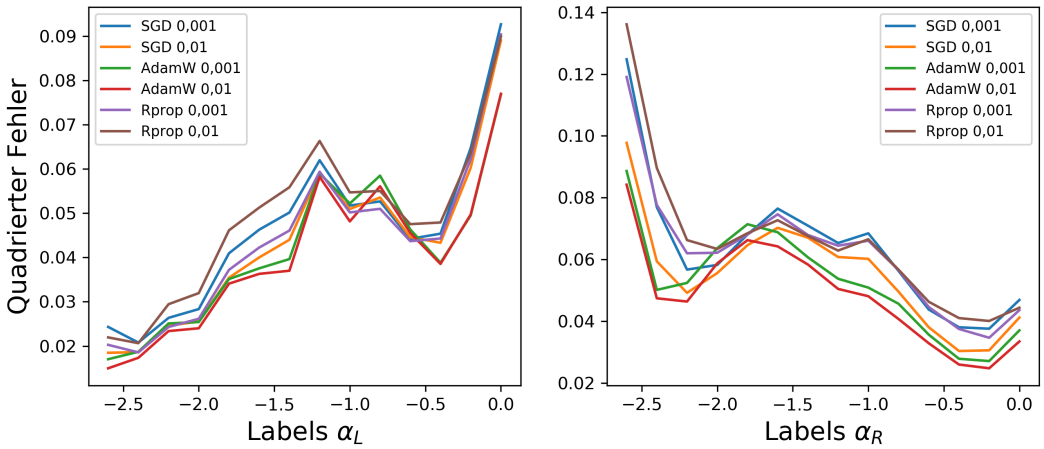


Abbildung 4.10: Verlauf der quadrierten Fehler bei verschiedenen Optimizern und Lernraten

4.7 Fazit

Insgesamt ergibt sich ein optimales Netz, wenn mindestens 9 Convolutional Layer mit gleichverteilten Pooling Layern und einer jeweils sich verdoppelten Neuronenzahl verwendet werden. Außerdem sollten ein Scheduler, Batch-Normalization, sowie AdamW als Optimizer (mit einer Start-Lernrate von $0,01$) Anwendung finden. Die Wahl der Aktivierungsfunktion scheint in der Praxis eine weniger bedeutende Rolle zu spielen. Theoretisch überzeugt aber SELU durch ihre nicht-sättigenden und selbst normalisierenden Eigenschaften, daher wird diese zur Berechnung der Parameter verwendet. Die Datasets werden zusätzlich auf einen Mittelwert von null und eine Standardabweichung von eins normiert.
Der finale Python Code des CNNs ist im Appendix aufgeführt.

5 Ergebnisse

5.1 Evaluation der Vorhersagen der CNNs

Für alle vier räumlichen Verteilungsmodelle mPWN, mSNR, mSp4 und mSp2b wurden für je beide Methoden Datasets berechnet und mit diesen das in Kapitel 4 beschriebene CNN trainiert. Dabei wurden die Datasets jeweils dreimal zufällig umsortiert, um durch den Zufall bedingte Unterschiede in den Rekonstruktionen der Parameter aus den HGPS Daten abschätzen zu können. Die Ergebnisse dieser CNN-Modelle, d.h. die trainierten CNNs, sind im Folgenden aufgeführt. Tabelle 5.1 zeigt die mittleren quadrierten Fehler aller CNN-Modelle, wobei für Methode 2 zusätzlich nur die Fehler im Parameterbereich von -3 bis 0 aufgeführt sind, um beide Methoden besser vergleichbar zu machen.

	Methode 1 Parameter von -3 bis 0		Methode 2 Parameter von -3 bis 0		Methode 2 Parameter von -3 bis 2	
	MSE α_L	MSE α_R	MSE α_L	MSE α_R	MSE α_L	MSE α_R
mPWN	0,05794	0,14470	0,03851	0,05093	0,1029	0,05579
	0,05837	0,14552	0,03932	0,05164	0,10533	0,05578
	0,05726	0,14608	0,03891	0,05135	0,10453	0,05643
mSNR	0,05630	0,14095	0,03250	0,04916	0,09072	0,05569
	0,05639	0,14403	0,03291	0,04898	0,09118	0,05606
	0,05630	0,14334	0,03249	0,04866	0,09109	0,05565
mSp4	0,05409	0,14401	0,02905	0,06116	0,0826	0,05837
	0,05472	0,14253	0,02895	0,06072	0,08215	0,0591
	0,05443	0,14193	0,02901	0,06079	0,08234	0,05977
mSp2b	0,06280	0,16364	0,03038	0,04623	0,07564	0,05598
	0,06182	0,16389	0,03053	0,04519	0,07361	0,05634
	0,06212	0,16446	0,03006	0,04586	0,07377	0,05555

Tabelle 5.1: Mittlerer quadrierter Fehler für alle CNN-Modelle

5.1.1 Methode 1: Quellen bekannter Ausdehnung und Entfernung

Die Vorhersagen von CNN-Modellen, welche mit Datasets nach Methode 1 trainiert wurden, entsprechen im Mittel relativ gut dem jeweiligen Vorgabe- bzw. Label-Parameter. Abbildung 5.1 zeigt für das erste CNN-Modell, welches auf die räumliche Verteilung von mPWN trainiert wurde, ein zweidimensionales Histogramm. In diesem ist die Häufigkeit verschiedener Label- und rekonstruierten Parameter-Paare aufgetragen. Dabei markiert die schwarz gestrichelte Linie den gewünschten Verlauf, bei dem der rekonstruierte Parameter seinem Label exakt entspricht. Zu erkennen ist, dass die Vorhersagen grob dem erwünschten Verlauf folgen. An den Rändern bei ungefähr $-2,8$ und 0 weichen die Vorhersagen dann im Mittel eher ab. Außerdem streuen die Vorhersagen für α_R deutlich mehr als für α_L. Das zeigt auch Abbildung

5.2, in der die Häufigkeit verschiedener Fehler (in Bins der Größe 0, 1) aufgetragen ist. In rot ist auch eine Linie aufgetragen, die den Erwartungswert markiert; außerdem zwei gepunktete Linien, die die Standardabweichung markieren. Beide Größen sind auch nochmal im Titel vermerkt. Diese Verteilungen sind auch bei den anderen CNN-Modellen ähnlich. Tabelle 5.1 zeigt für jedes CNN-Modell die gemittelten quadrierten Fehler.

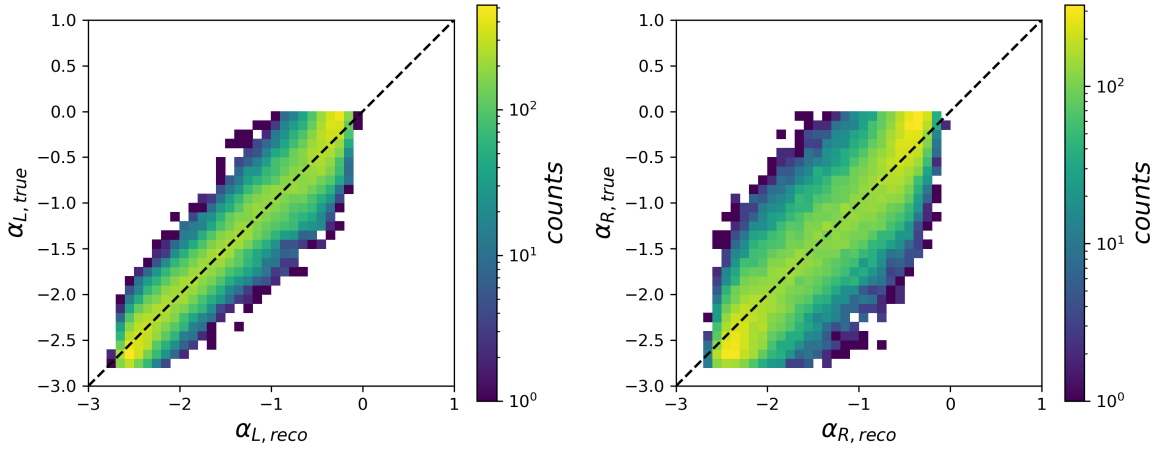


Abbildung 5.1: 2D-Histogramm der Label- und rekonstruierten Parameter (Methode 1)

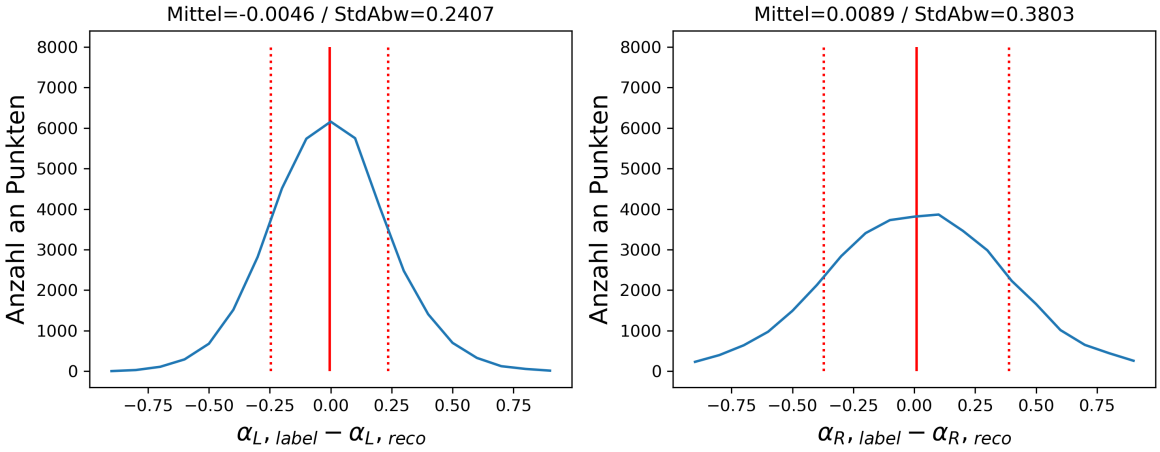


Abbildung 5.2: Die Häufigkeit von Vorhersagenfehler (Differenz von Label und rekonstruiertem Parameter), trainiert mit mPWN

5.1.2 Methode 2: Alle Quellen

Auch für Vorhersagen vom ersten CNN-Modell, das unter Methode 2 mit mPWN operierte, sind in Abbildung 5.3 die Vorhersagen als Histogram aufgetragen. Abbildung 5.4 zeigt wie schon für Methode 1 die Häufigkeitsverteilung der Fehler. Vergleicht man diese und die gemittelten quadrierten Fehler in Tabelle 5.1 mit denen der Methode 1, fallen mehrere Dinge auf: Für α_L ist der MSE unter Methode 1 (auf dem dort angewandten Parameterbereich von -3 bis 0) geringer, als auf dem größeren Parameterbereich unter Methode 2. Betrachtet man

allerdings für Methode 2 den gleichen Parameter-Bereich wie in Methode 1, ist der MSE diesmal bei Methode 2 geringer. Wie auch Abbildung 5.3 zeigt, sind die Rekonstruktionen von positiven α_L deutlich fehleranfälliger als die von negativen. Auch für die Rekonstruktionen von α_R besteht dieser Effekt, allerdings weniger auffällig. Hier ist in beiden Fällen Methode 2 zuverlässiger als Methode 1, da die Ergebnisse weniger streuen.
Die in Kapitel 3 beschriebenen Abschätzungen unbekannter Parameter von den übrigen Quellen scheinen damit sinnvoll zu sein, da dem CNN mehr Informationen für verlässlichere Vorhersagen zur Verfügung stehen.

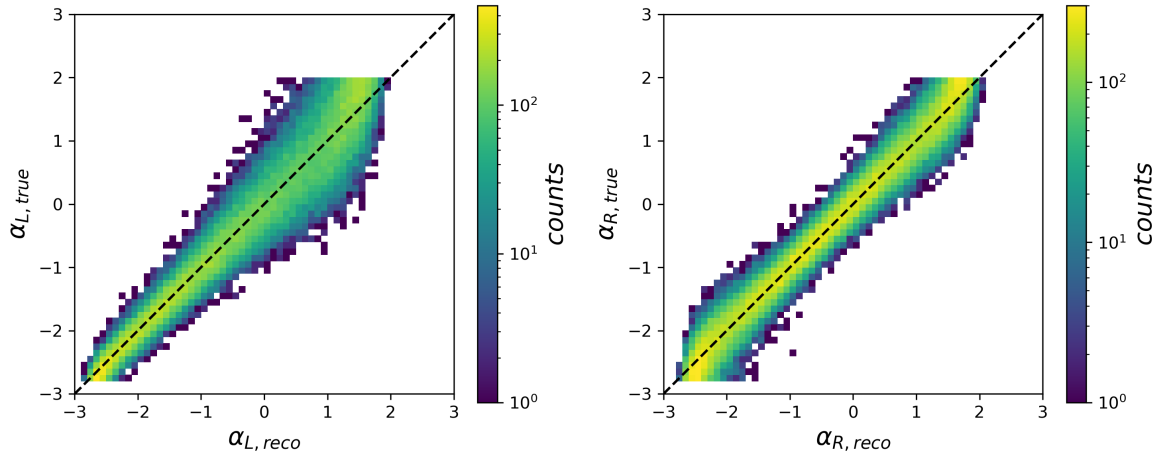


Abbildung 5.3: 2D-Histogramm der Label- und rekonstruierten Parameter (Methode 2)

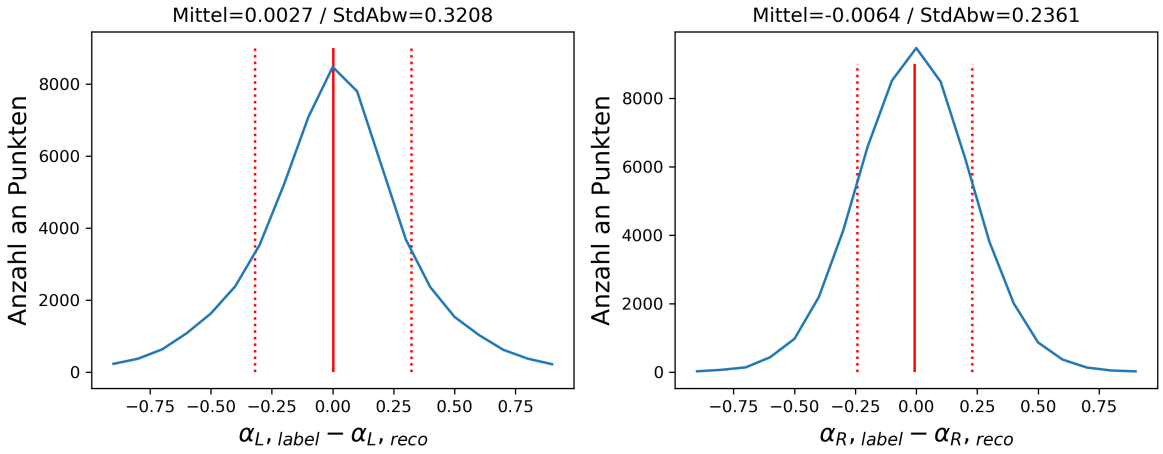


Abbildung 5.4: Die Häufigkeit von Vorhersagenfehler (Differenz von Label und rekonstruiertem Parameter), trainiert mit mPWN Modell

5.2 Rekonstruktionen anhand der HGPS Quellen

Tabelle 5.2 zeigt die „realen“, anhand der HGPS-Daten rekonstruierten Parameter. Für Methode 1 sind nur die 16 Quellen bekannter Ausdehnung und Entfernung des HGPS-Katalogs

verwendet worden, für Methode 2 dagegen alle 68 Quellen, deren Fluss die jeweilige Sensitivität überschreitet.

	Methode 1		Methode 2	
	α_L	α_R	α_L	α_R
mPWN	-2,1494240	-0,22548789	-2,421889	1,0080618
	-2,1850624	-0,2631985	-2,4877744	0,9466509
	-2,2786584	-0,24210393	-2,4356253	0,90213597
mSNR	-2,0800362	-0,28089976	-2,464862	0,62412786
	-2,154513	-0,2700755	-2,4083338	0,5825676
	-2,1422534	-0,1984514	-2,2719817	1,071278
mSp4	-1,9916496	-0,22814047	-2,3016596	0,7735214
	-2,1108057	-0,3268782	-2,2878788	0,6507912
	-2,0670385	-0,35443735	-2,3521595	0,6462418
mSp2b	-2,0515127	-0,30587775	-2,3335147	0,61029327
	-2,0008988	-0,19638407	-2,250733	0,49404532
	-2,1043305	-0,22759914	-2,3657544	0,5469477

Tabelle 5.2: Rekonstruierte Verteilungsparameter für alle CNN-Modelle

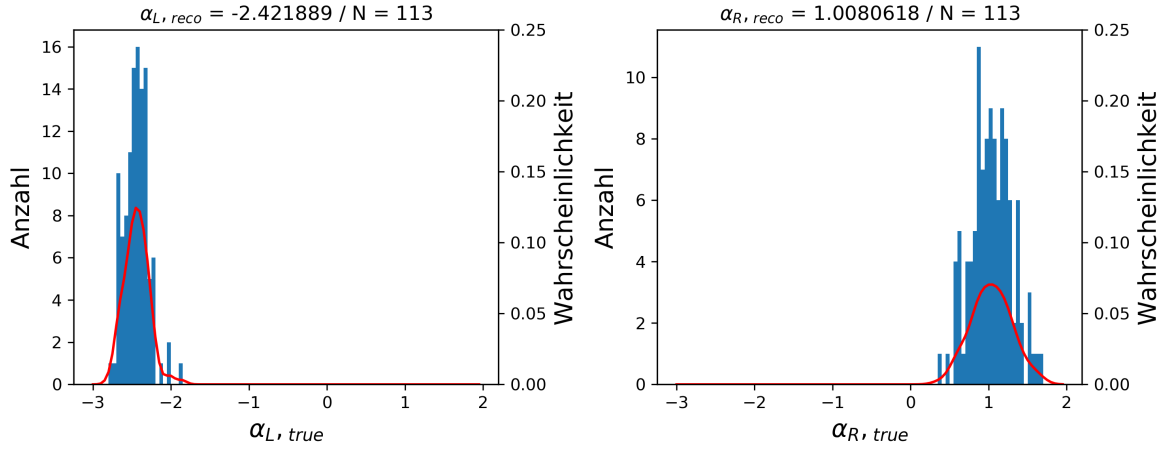
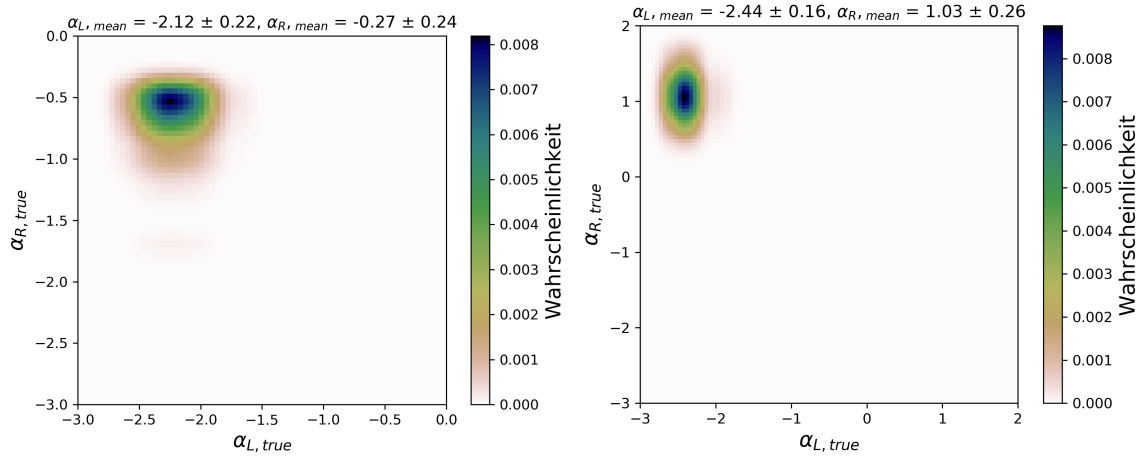


Abbildung 5.5: Aus den Vorhersagen simulierter Daten entnommene Verteilung der Label-Parameter für die rekonstruierten Parameter (Methode 2), in blau: Histogramme, in rot: KDE-Wahrscheinlichkeitsverteilung

Mit den Vorhersagen aus den simulierten Validierungs-Daten lassen sich nun bedingte Wahrscheinlichkeitsverteilungen, für gegebene rekonstruierte Parameter ermitteln. Das bedeutet die Wahrscheinlichkeit verschiedener „realer“ Parameter bei festen rekonstruierten Parametern lässt sich damit abschätzen. Abbildungen 5.5 zeigt beispielhaft für das erste mPWN CNN-Modell (Methode 2) in blau Histogramme und in rot die Wahrscheinlichkeitsdichte.

Die Histogramme ergeben sich aus der Anzahl an Label-Parametern, deren vorhergesagten α_L-α_R Paare den „wahren“, aus den HGPS-Daten rekonstruierten Parametern, ungefähr $(\pm 0,1)$ entsprechen. Die Anzahl an jenen Paaren ist im Titel durch N gegeben. Die rote Kurve ist eine gauss'sche Kernel-Density-Estimation, d.h. die Dichte an Datenpunkten



(a) (b)

Abbildung 5.6: Die zweidimensionale Wahrscheinlichkeitsverteilung (KDE) der rekonstruierten Parameter beider Methoden

in 0,1 großen Kernels jeweils als Gaussverteilung dargestellt, welche aufsummiert und anschließend als Wahrscheinlichkeitsverteilung normiert wurden. Abbildung 5.6 zeigt die KDE-Wahrscheinlichkeitsverteilungen auch nochmal für beide Parameter kombiniert; jeweils für Methode 1 und 2. Die KDE-Verteilungen für alle CNN-Modelle sind im Appendix aufgeführt. Tabelle 5.3 führt die Erwartungswerte mit ihrer Standardabweichung, welche sich aus der KDE-Wahrscheinlich- keitsverteilung ergeben.

	Methode 1		Methode 2		Paper [13]	
	α_L	α_R	α_L	α_R	α_L	α_R
mPWN	-2,112 ± 0,219	-0,272 ± 0,237	-2,439 ± 0,156	1,035 ± 0,259	-1,81	-1,13
	-2,188 ± 0,214	-0,291 ± 0,223	-2,527 ± 0,148	0,922 ± 0,238		
	-2,259 ± 0,184	-0,296 ± 0,233	-2,460 ± 0,147	0,894 ± 0,265		
mSNR	-2,048 ± 0,217	-0,335 ± 0,273	-2,500 ± 0,165	0,630 ± 0,232	-1,70	-1,19
	-2,158 ± 0,233	-0,315 ± 0,239	-2,424 ± 0,177	0,570 ± 0,230		
	-2,122 ± 0,202	-0,284 ± 0,246	-2,335 ± 0,160	1,104 ± 0,303		
mSp4	-2,032 ± 0,223	-0,338 ± 0,272	-2,360 ± 0,161	0,773 ± 0,274	-1,64	-1,17
	-2,029 ± 0,192	-0,241 ± 0,191	-2,300 ± 0,188	0,639 ± 0,245		
	-2,086 ± 0,224	-0,273 ± 0,238	-2,392 ± 0,158	0,652 ± 0,252		
mSp2b	-1,980 ± 0,233	-0,240 ± 0,199	-2,340 ± 0,181	0,615 ± 0,242	-1,78	-1,62
	-2,114 ± 0,242	-0,363 ± 0,280	-2,265 ± 0,183	0,459 ± 0,205		
	-2,059 ± 0,240	-0,410 ± 0,331	-2,358 ± 0,174	0,518 ± 0,230		

Tabelle 5.3: Erwartungswert mit Standardabweichung für die rekonstruierten Verteilungsparameter anhand der Wahrscheinlichkeitsverteilung der Label-Parameter aus den Validierungsdaten aller CNN-Modelle, zum Vergleich die Ergebnisse aus [13]

5.3 Diskussion

Betrachtet man die Ergebnisse in Tabelle 5.3 und vergleicht sie miteinander und mit denen aus [13], fallen vor allem zwei Dinge auf: Zum einen unterscheiden sich die rekonstruierten Parameter zwischen Methode 1 und Methode 2 und zum anderen weichen die Parameter stark von denen aus [13] ab.

Die Unterschiede zwischen den Methoden
Die rekonstruierten Luminositätsparameter α_L unterscheiden sich eher gering voneinander und ihre jeweiligen Unsicherheitsbereiche überschneiden sich. Dagegen unterscheiden sich die Radiusparameter α_R deutlich stärker. Da dieser unter Methode 2 sogar positiv wurde und damit den ursprünglichen Parameterbereich von -3 bis 0 verlassen hatte, musste der Bereich erweitert werden. Die Extrapolation des CNNs außerhalb der Grenzen des Bereichs war höchst unzuverlässig. Grund für die Unterschiede könnte die Annahme sein, zufällig zwei Drittel aller simulierten Quellen in Methode 2 als Quellen mit unbekannter Entfernung zu behandeln. Eventuell sind die Entfernungen von Quellen bestimmter Klassen besser abschätzbar, so dass für diese zu einer höheren Wahrscheinlichkeit Informationen über die Entfernung existieren. Dadurch könnte das zufällige Auswählen von simulierten Quellen unbekannter Entfernung die realen Verhältnisse nur ungenügend beschreiben. Ein anderer Grund kann sein, dass der Anteil von Quellen bekannter Ausdehnung bzw. der Anteil von Punktquellen in den simulierten Populationen stark varriiert. Während bei Methode 1, so wie im HGPS-Katalog, jede Population stets aus 16 Quellen bekannter Ausdehnung und Entfernung bestand, liegt der Anteil ausgedehnter Quellen unter Methode 2 durchschnittlich bei 55% mit einer Standardabweichung von 30%. In Einzelfällen gibt es sogar Populationen, die ausschließlich bzw. auch überhaupt nicht aus ausgedehnten Quellen bestehen. Der Anteil ausgedehnter Quellen im HGPS-Katalog liegt bei 78%.

Die Unterschiede zwischen Machine-Learning und der Maximum Likelihood Methode aus [13]
Wie bereits in Kapitel 3 erwähnt, wurden in [13] die Parameter über eine Maximum Likelihood Methode bestimmt. Sowohl die durch den Machine-Learning Ansatz erhaltenen Ergebnisse für den Luminositäts- als auch für den Radiusparamter unterscheiden sich von denen aus [13]. Das legt den Schluss nahe, dass die verwendeten Modellannahmen so noch zu einfach sind.
Auch hier könnte ein Teil der Unterschiede durch die oben bereits beschriebenen, anderen Verhältnisse der Anzahl ausgedehnter Quellen zur Gesamtzahl von Quellen einer Population bedingt sein. In [13] lag dieser Anteil zwischen 23% und 38%.
Ein weiter Grund liegt wohlmöglich in den räumlichen Modellen. Abbildung 5.7 zeigt beispielsweise erneut ein zweidimensionales Histogramm der rekonstruierten und Label-Parameter Paare. Allerdings zeigt sie das für ein CNN-Modell, das zwar auf das räumliche Modell mPWN trainiert, jedoch mit dem Validierungsset von Quellpopulationen, die nach mSNR verteilt sind, getestet wurde. Man erkennt, dass die Vorhersagen verzerrt sind. Das bedeutet, dass auch kleinere Abweichungen zwischen der räumlichen Verteilung der Quellen mit der trainiert wurde und der, auf die getestet wurde, zusätzliche Ungenauigkeiten in die Rekonstruktion der Parameter bringen.
Eine weitere Möglichkeit liegt in der Annahme, dass die Verteilung von Radien und Luminositäten der galaktischen Gammaquellen einer einfachen Potenzverteilung folgen. Die Vorgänge in Gammaquellen könnten komplexer und unterschiedlicher sein und die Entwicklung der Aus-

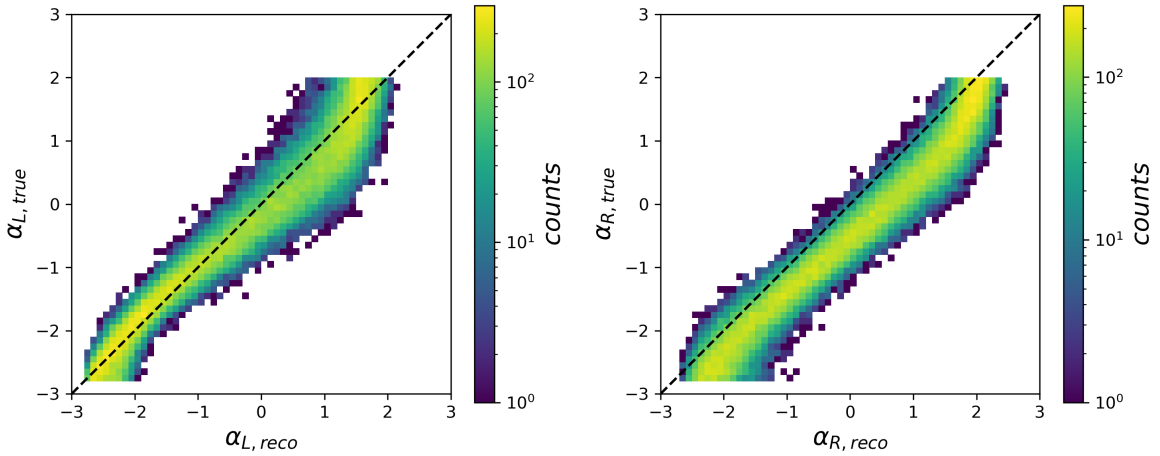


Abbildung 5.7: 2D-Histogramm der Label- und rekonstruierten Parameter eines CNNs, dass auf mPWN trainiert, aber mit nach mSNR verteilten Populationen getestet wurde

dehnung bzw. Luminosität einer Gammaquelle könnte demnach im Laufe ihres Lebens ist viel dynamischer sein als eine Potenzverteilung abbildet. So expandieren z.B. junge, ungefähr einige tausend Jahre alte, Supernovaüberreste (SNR) und Pulsarwindnebel (PWN) zunächst schneller und später etwas langsamer [26, 27]. Dies könnte die positiven Radius-Parameter unter Methode 2 erklären. Geht man davon aus, dass die Mehrheit der Gammaquellen aus SNRs und PWNs besteht, wobei im HGPS-Katalog mindestens 8 Quellen SNRs sind, und mindestens 10 Quellen PWNs, so müsste ein Abbild vieler Quellen in unterschiedlichen Stadien ihrer Entwicklung eben jene Verteilungen widerspiegeln. Diese würden sich durch einen positives α_R kennzeichnen. Natürlich muss diese Expansion irgendwann abbrechen, wie auch in [27] vermutet wird. Ein Sample bestehend aus Quellen in diesem Entwicklungsstadium, mindestens mehrere Zehntausend Jahre alt [27], würde wiederum eine negatives α_R bedeuten. Für die Entwicklung der Leuchtkraft gilt, dass in den ersten hundert bis tausend Jahren des Lebens möglicher Gammaquellen (der Phase mit zwei Schocks bei SNRs bzw. der Phase der freien Expansion bei PWNs) die Luminosität rasant ansteigt, jedoch im Verlauf des restlichen Lebens wieder langsam abfällt [27, 28]. Da diese Phase deutlich länger andauert, ist zu vermuten, dass auch mehr Quellen in späteren Entwicklungsstadien detektiert wurden. Das hätte einen negativen Luminositätsparameter, wie er auch rekonstruiert wurde, zur Folge.
Es sollte nicht vergessen werden, dass PWNs und SNRs nicht ausschließlich alle Gammaquellen abbilden und auch der Einfluss anderer Objekte vielleicht ebenfalls eine Rolle spielen, warum eine Potenzverteilung wohlmöglich nicht die realen Verhältnisse widerspiegelt.

5.4 Ausblick

Nichtsdestotrotz konnten im Rahmen dieser Annahmen Parameter rekonstruiert werden. Dabei wurde festgestellt, dass die Abschätzungen von Luminosität und Radius simulierter Gammaquellen unbekannter Entfernung und/oder Ausdehnung sinnvoll sind, da so im Training bessere Vorhersagen getroffen werden konnten. Um möglichen Ursachen von Ungenauigkeiten zu begegnen, ließe sich der Machine-Learning Ansatz noch dahingehend ausbauen, dass man den Quellen auch entsprechende Klassen (Supernova Überreste, Pulsarwindnebel, binäres Sternsystem, ...) entsprechend ihrer Häufigkeit im HGPS-Katalog zuordnet und anschlie-

ßend individuelle Verteilungen modelliert. Sowohl individuelle räumliche Verteilungen (da sich nicht zuletzt auch die Modelle mPWN und mSNR an den Verteilungen von PWNs und SNRs orientieren), als auch individuelle Verteilungen von Radius und Luminosität entsprechend ihrer Entwicklung im Laufe des Lebens der Quelle einer bestimmten Klasse, scheinen lohnenswerte Ansätze zu sein. Dabei müssten die Verteilungen für Quellen, die bisher keiner Klasse zugeordnet werden konnten, entweder vorerst vernachlässigt oder andersweitig abgeschätzt werden. Auch eine Anpassung in der Simulation der Populationen dahingehend, dass die Anteile an ausgedehnten Quellen und Punktquellen eher auch denen des HGPS entsprechen, kann ein vielversprechender Versuch sein, zuverlässige Ergebnisse zu rekonstruieren. Neben erweiterten Modellannahmen lohnt sich bestimmt auch eine genauere Analyse der Methodiken und Ergebnisse des Machine-Learning-Ansatzes und der Maximunm Likelihood Methode, um noch weitere Informationen und Ideen zur Rekonstruktion von Luminosität und Radius zu erhalten.

6 Zusammenfassung

Der H.E.S.S. Galactic Plane Survey Katalog diente als Grundlage zur Rekonstruktion der Verteilung von Luminosität und Radius jener galaktischer Gammaquellen. Zu beachten war dabei vor allem ein großer Beobachtungsbias, so dass die detektierten Quellen nicht repräsentativ für die Gesamtheit an Gammaquellen in der Milchstraße sind. Es wurde versucht, dem Beobachtungsbias über Machine-Learning zu begegnen. Dafür wurden künstliche Gammaquellpopulationen simuliert, unter Einbezug mehrerer möglicher räumlichen Verteilungen. Aus diesen synthetischen Quellen wurden unter der Verwendung zwei verschiedener Methoden Datasets berechnet. Zum einen wurden in der ersten Methode die synthetischen Populationen so erstellt, dass jeweils 16 Quellen mit bekannter Ausdehnung und Entfernung gemessen werden konnten. Zum anderen bestand die zweite Methode darin die Populationen so zu erstellen, dass allgemein 68 Quellen detektiert werden können und fehlende Informationen zur Ausdehnung und Entfernung mancher Quellen stochastisch abgeschätzt wurde.
Unter der Annahme, dass Luminosität und Radius von Gammaquellen voneinander unabhängig potenzverteilt sind, wurde die Datasets dann genutzt um mithilfe eines neuralen Netzes die Modellparameter α_L und α_R zu rekonstruieren, welche die Verteilungsfunktion maßgeblich bestimmen. Es wurde ein Convolutional Neural Network (CNN) trainiert und optimiert. Dafür wurden verschiedene Parameter wie die Layer-Anzahl, die Initialisierung der Gewichtungen, verwendete Aktivierungsfunktionen oder auch der verwendete Optimizer variiert und optimiert. Anschließend wurden verschiedene Ausführungen des CNNs anhand der unterschiedlichen Datasets erstellt, evaluiert und zur Rekonstruktion der realen Verteilungsfunktionen genutzt.
Der Fehler von Vorhersagen von CNN-Modellen der Methode 1 lag, wie bei CNN-Modellen der Methode 2, im Mittel bei null. Allerdings streuten die Vorhersagen von Models der Methode 1 stärker, das den Schluss nahelegt, dass die stochastischen Abschätzungen sinnvoll sind, da so zuverlässigere Ergebnisse erzielt werden konnten.
Die aus dem HGPS-Katalog rekonstruierten und mittels einer Kernel-Density-Estimation abgeschätzten Parameter lagen für Methode 1 (also nur unter der Berücksichtigung von Quellen bekannter Ausdehnung und Entfernung) für die Luminosität je nach verwendetem Raummodell zwischen -2.2 und $-2,0$ und für den Radius zwischen $-0,24$ und $-0,3$. Bei Methode 2 (also unter Berücksichtigung aller Quellen, wobei unbekannte Informationen stochastisch abgeschätzt wurden) für die Luminosität zwischen $-2,5$ und $-2,3$ und für den Radius zwischen $0,5$ und $1,00$.
Verschiedene Modellannahmen könnten noch zu ungenau für die realen Verhältnisse sein. Darunter fallen, dass Quellen unbekannter Entfernung zufällig bestimmt werden oder Radien und Luminositäten galaktischer Gammaquellen einer einfachen Potenzverteilung folgen. Zusätzliche Unsicherheiten über die räumliche Verteilung der Quellen, sowie stark variierende Anteile von ausgedehnten Quellen in simulierten Populationen, könnten weitere Ungenauigkeiten verursachen. Deshalb bietet es sich zukünftig an hier den Machine-Learning-Ansatz noch weiter auszubauen.

7 Appendix

Python Code des CNN

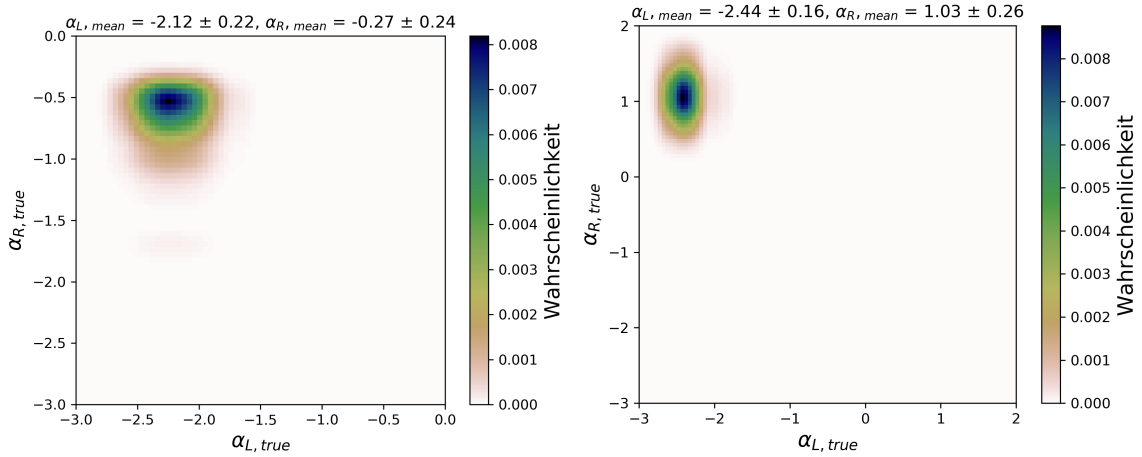
Listing 7.1: Python Code des CNN

```
from multiprocessing import cpu_count
from decimal import *
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
import numpy as np
import h5py
import matplotlib.pyplot as plt
import matplotlib.colors as cls
from tqdm.notebook import tqdm
import torch
from torch.utils import data
import torch.nn.functional as F
import torch.nn as nn
import torch.utils.data
import torch.optim as optim
from torch.autograd import Variable
import torch.optim.lr_scheduler as scheduler
import torchvision

def H5_Dataset(filepath):
    with h5py.File(filepath, 'r') as f:
        data = torch.from_numpy(np.array(f['data'])).type(torch.float)
        label = torch.from_numpy(np.stack([np.array(f['alpha_L']).ravel(),
                                           np.array(f['alpha_R']).ravel()
                                          ],
                                          axis=1
                                         )
                                ).type(torch.float)
    return torch.utils.data.TensorDataset(data, label)

def h5_to_dataloader(filepath, batch_size, shuffle=True, num_workers=cpu_count()):
    dataloader = torch.utils.data.DataLoader(H5_Dataset(filepath),
                                             batch_size=batch_size,
                                             shuffle=shuffle,
                                             num_workers=num_workers
                                            )
    return dataloader

params = {'batch_size': 128,
          'shuffle': True,
          'num_workers': cpu_count()}

training_dataloader = h5_to_dataloader('../datasets/training_set.h5', **params)
validation_dataloader = h5_to_dataloader('../datasets/validation_set.h5', **params)

class conv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size):
```

```
        super(conv, self).__init__()
        self.ops = nn.Sequential(nn.Conv2d(in_channels,
                                           out_channels,
                                           kernel_size,
                                           stride=1,
                                           padding=1,
                                           bias=False),
                                 nn.BatchNorm2d(out_channels),
                                 nn.SELU())
    def forward(self, input):
        x = self.ops(input)
        return x

class fc(nn.Module):
    def __init__(self, in_features, out_features):
        super(fc, self).__init__()
        self.ops = nn.Sequential(nn.Linear(in_features,
                                           out_features,
                                           bias=True
                                          ),
                                 nn.BatchNorm1d(out_features),
                                 nn.ELU())
                                 #nn.PReLU(out_features,0.05))
    def forward(self, input):
        return self.ops(input)

class Net(nn.Module):
    def __init__(self, name='Net'):
        super(Net, self).__init__()
        self.name = name
        self.conv = nn.Sequential(conv(1,2,3),
                                  conv(2,4,3),
                                  conv(4,8,3),
                                  nn.AvgPool2d(2),
                                  conv(8,16,3),
                                  conv(16,32,3),
                                  conv(32,64,3),
                                  nn.AvgPool2d(2),
                                  conv(64,128,3),
                                  conv(128,256,3),
                                  conv(256,512,3),
                                  nn.AvgPool2d(2)
                                 )

        self.fc = nn.Sequential(nn.Dropout(0.5),
                                nn.Linear(512,2))

        nn.init.xavier_normal_(self.conv[0].ops[0].weight)
        nn.init.xavier_normal_(self.conv[1].ops[0].weight)
        nn.init.xavier_normal_(self.conv[2].ops[0].weight)
        nn.init.xavier_normal_(self.conv[4].ops[0].weight)
        nn.init.xavier_normal_(self.conv[5].ops[0].weight)
        nn.init.xavier_normal_(self.conv[6].ops[0].weight)
        nn.init.xavier_normal_(self.conv[8].ops[0].weight)
        nn.init.xavier_normal_(self.conv[9].ops[0].weight)
        nn.init.xavier_normal_(self.conv[10].ops[0].weight)

    def forward(self, input):
```

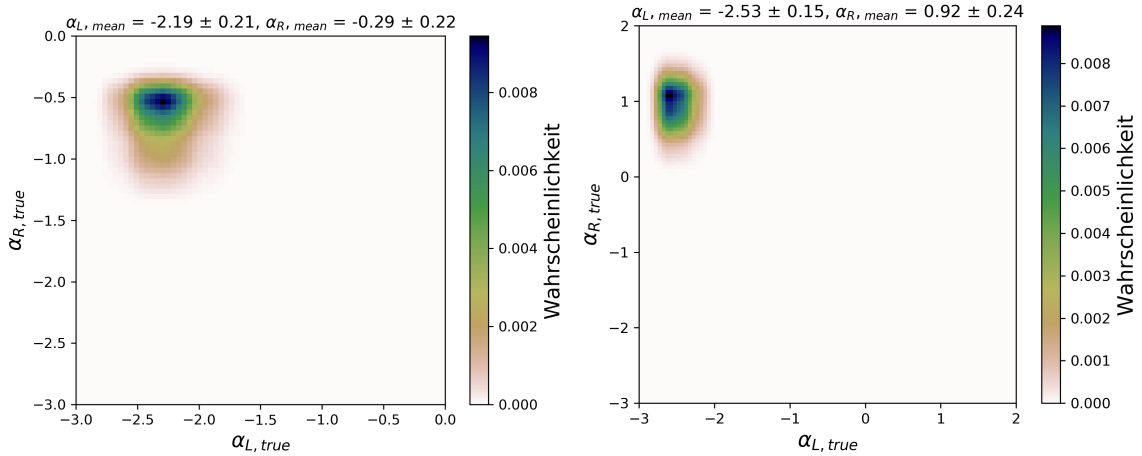
```
        x = self.conv(input)
        x = x.view(-1, 512)
        return self.fc(x)

class model:
    def __init__(self, train_dataloader, val_dataloader, net,
                 epochs=10, epochs_to_save=5):
        self.train_dataloader = train_dataloader
        self.val_dataloader = val_dataloader
        if torch.cuda.is_available():
            self.device = torch.device('cuda:0')
        else:
            self.device = torch.device('cpu')
        self.net = net
        self.epochs = epochs
        self.epochs_to_save = epochs_to_save
        self.valid_curve = -np.ones(epochs)
        self.train_curve = -np.ones(epochs)
        logfilename = self.net.name+'_progress.log'
        self.logger = logging.getLogger(self.net.name)
        self.logger.setLevel(logging.INFO)
        filehandler = logging.FileHandler(logfilename, 'w')
        filehandler.setFormatter(formatter)
        filehandler.setLevel(logging.INFO)
        self.logger.addHandler(filehandler)
        self.logger.addHandler(logging.StreamHandler())

    def train(self, lr = 0.01):
        nbts = 4
        criterion = nn.MSELoss()
        optimizer = optim.AdamW(self.net.parameters(), lr)
        self.tepoch = []
        self.tloss = []
        self.tlr = []
        self.vepoch = []
        self.vloss = []
        self.train_time = 0
        best_loss = 1e12
        self.scheduler = scheduler.ReduceLROnPlateau(optimizer, mode='max',
                                                     factor=0.5, patience=5,
                                                     verbose=False, threshold=1,
                                                     threshold_mode='abs',
                                                     min_lr=1e-10
                                                    )
        for epoch in tqdm(range(self.epochs)):
            for dataloader, net_phase, phase in zip([self.train_dataloader,
                                                     self.train_dataloader,
                                                     self.val_dataloader
                                                    ],
                                                    ['train',
                                                     'eval',
                                                     'eval'
                                                    ],
                                                    ['training',
                                                     'train_lc',
                                                     'valid_lc']
                                                    ):
                self.net.to(self.device)
```

```
            running_loss = 0.
            total_sum = 0.
            correct_sum = 0.
            if net_phase=='train':
                self.net.train()
                use_grad = True
            elif net_phase=='eval':
                self.net.eval()
                use_grad = False
            with torch.set_grad_enabled(use_grad):
                for i, data in enumerate(dataloader, 0):
                    inputs, labels = data
                    inputs, labels = (Variable(inputs).float().to(self.device),
                                      Variable(labels).float().to(self.device)
                                     )
                    optimizer.zero_grad()
                    outputs = self.net(inputs)
                    tloss = criterion(outputs, labels)
                    if net_phase=='train':
                        tloss.backward()
                        optimizer.step()
                    running_loss += tloss.detach().item()
                    total_sum += outputs.data.size()[0]
            loss = running_loss / total_sum
            learning_rate =  optimizer.param_groups[0]['lr']
            if phase == 'train_lc':
                self.train_curve[epoch] = loss
            if phase == 'valid_lc':
                self.scheduler.step(loss)
                self.valid_curve[epoch] = loss
                if loss < best_loss:
                    torch.save(self.net.to(torch.device('cpu')).state_dict(),
                               self.net.name+'.ptmodel')
        with h5py.File(self.net.name+'_learning_curves.h5', 'w') as h:
            h.create_dataset("epoch", data=np.arange(1,self.epochs+1))
            h.create_dataset("train", data=self.train_curve)
            h.create_dataset("valid", data=self.valid_curve)
```



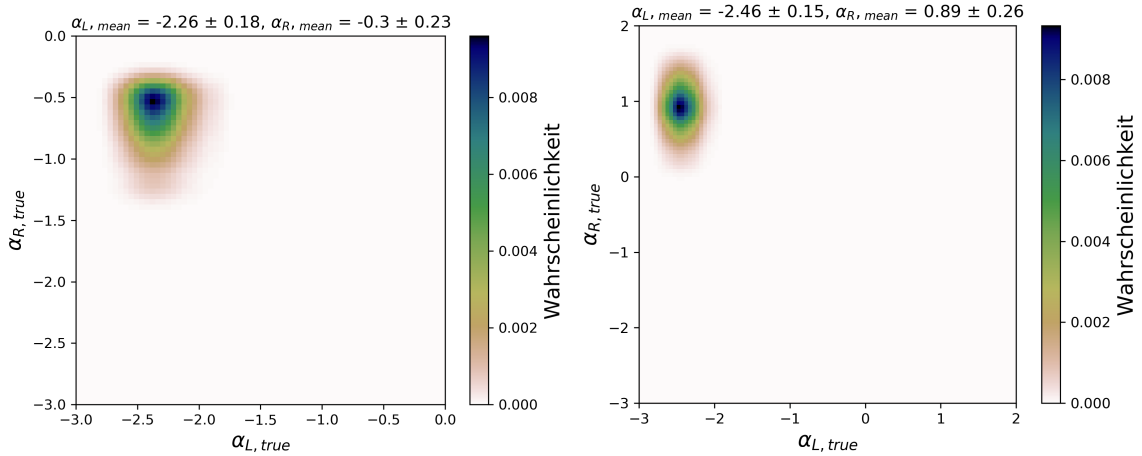
(a)

(b)



(c)

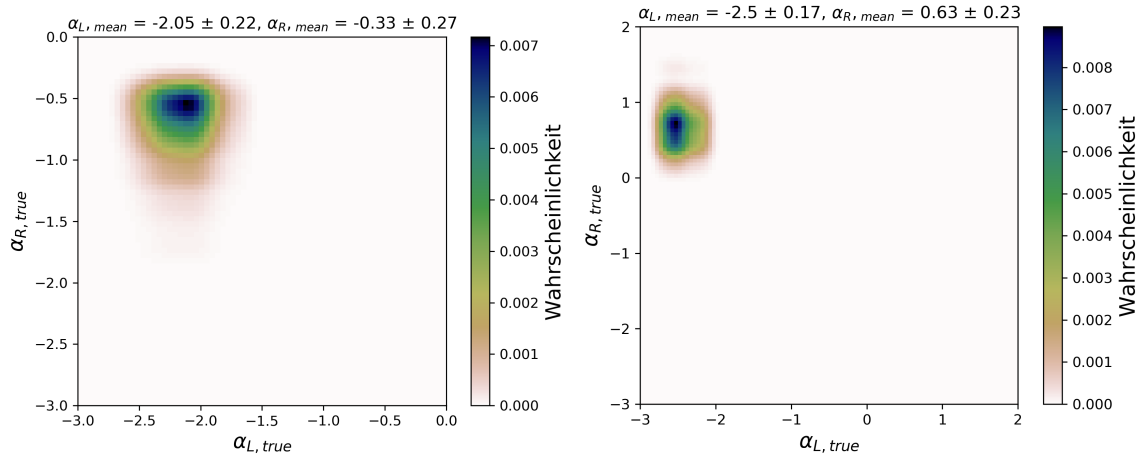
(d)



(e)

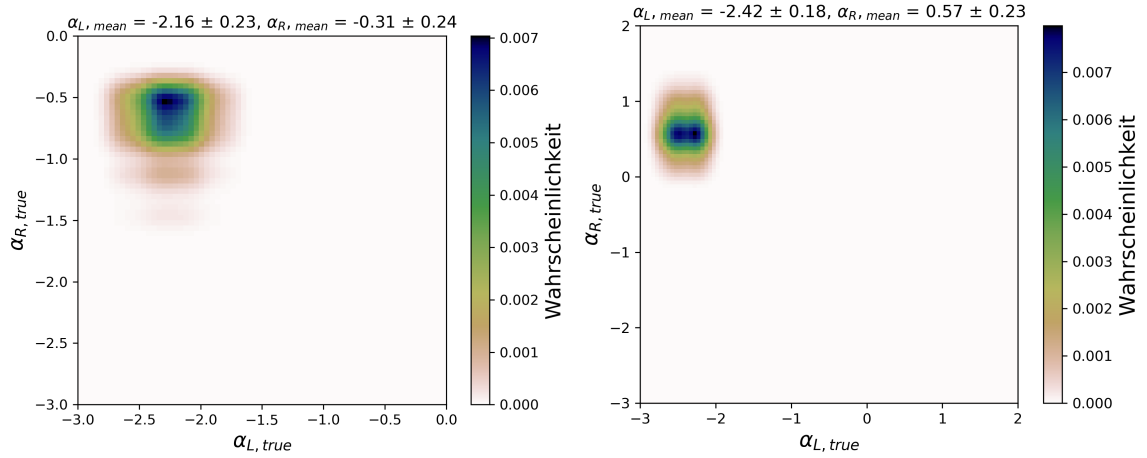
(f)

Abbildung 7.1: KDE-Plots mPWN, (a) (c) (e) Methode 1, (b) (d) (f) Methode 2



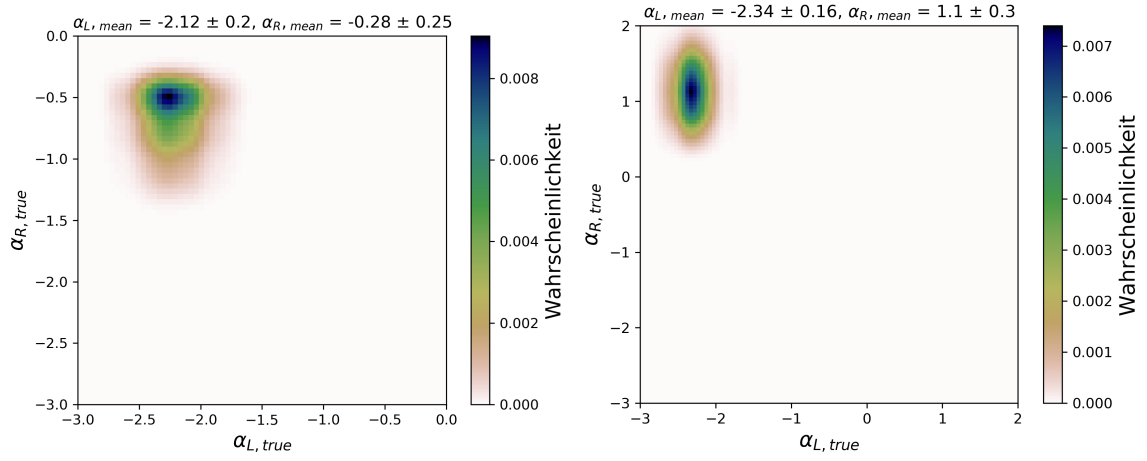
(a)

(b)



(c)

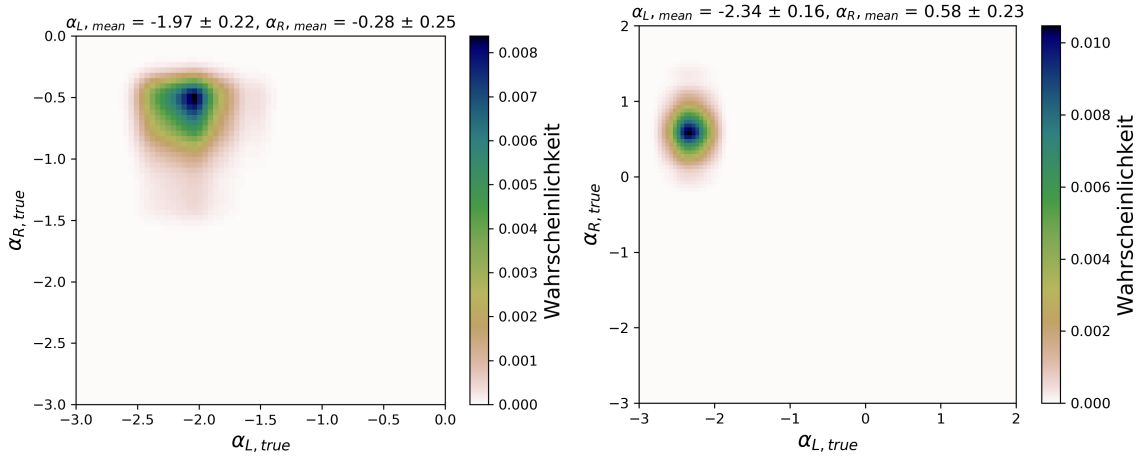
(d)



(e)

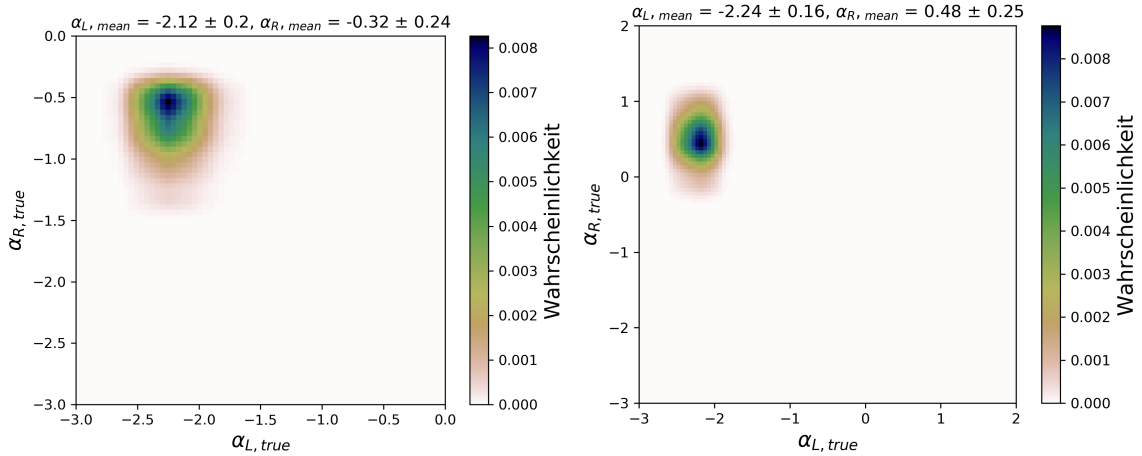
(f)

Abbildung 7.2: KDE-Plots mSNR, (a) (c) (e) Methode 1, (b) (d) (f) Methode 2



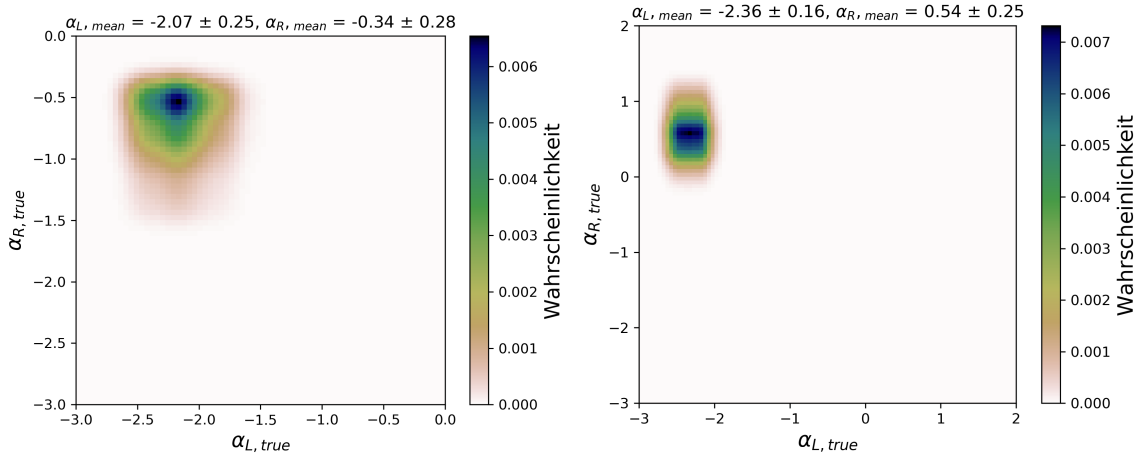
(a)

(b)



(c)

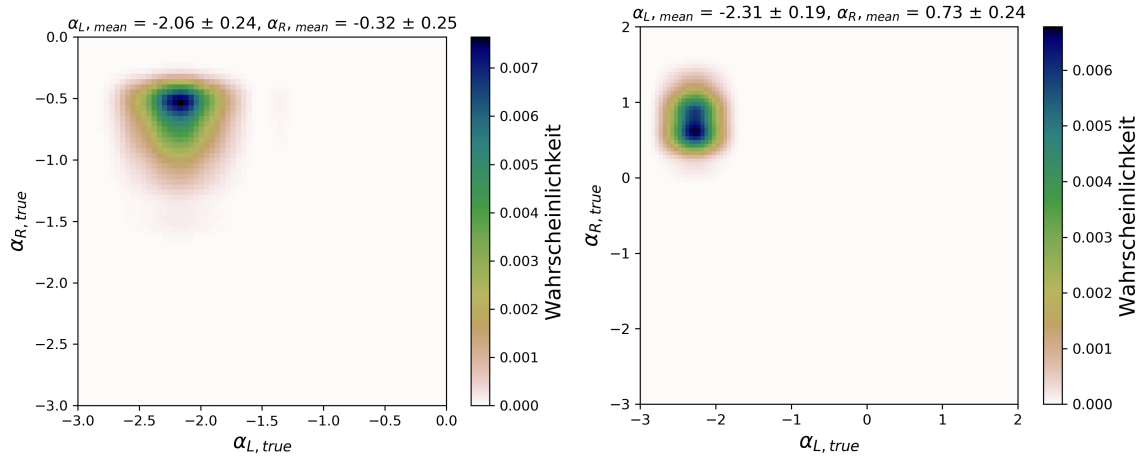
(d)



(e)

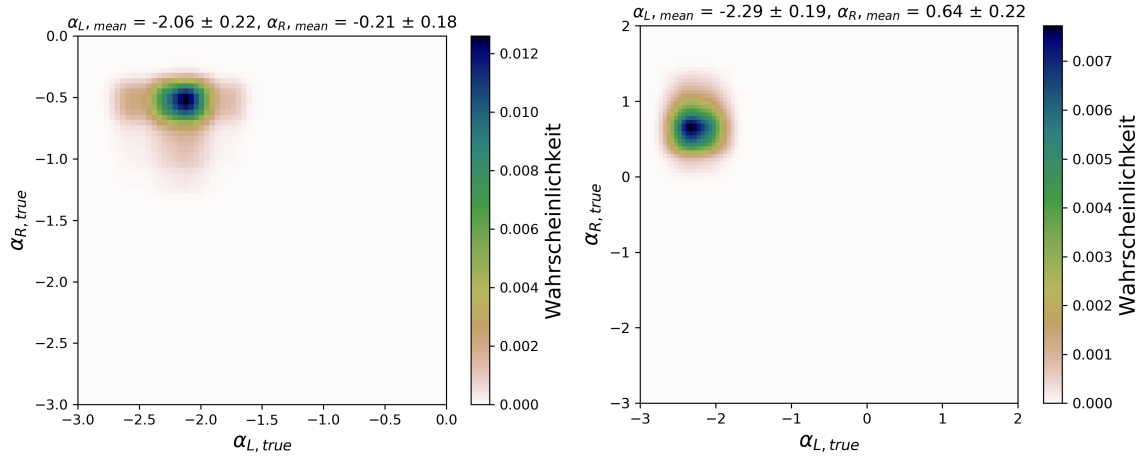
(f)

Abbildung 7.3: KDE-Plots mSp4, (a) (c) (e) Methode 1, (b) (d) (f) Methode 2



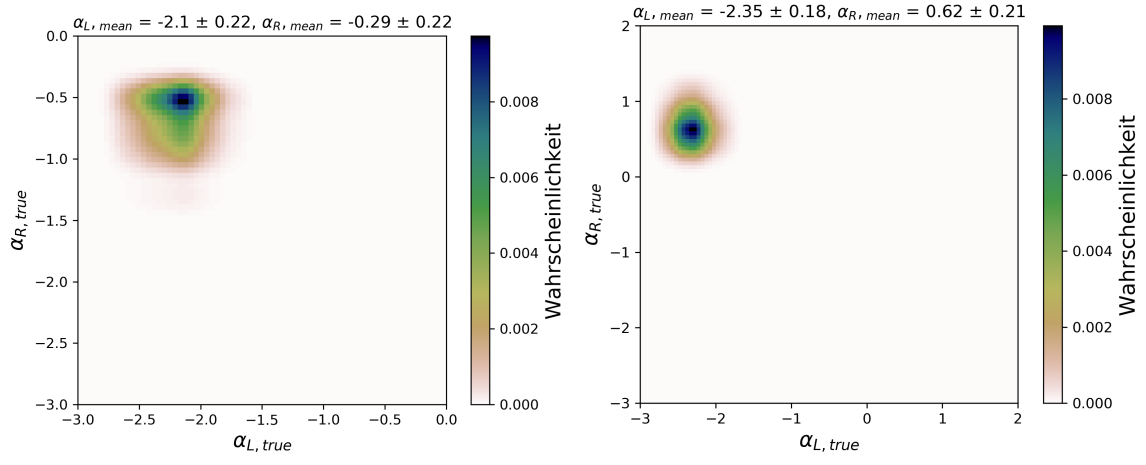
(a)

(b)



(c)

(d)



(e)

(f)

Abbildung 7.4: KDE-Plots mSp2b, (a) (c) (e) Methode 1, (b) (d) (f) Methode 2

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Claus Grupen. *Einstieg in die Astroteilchenphysik.* Springer Spektrum, Heidelberger Platz 3, 14197 Berlin, 2018.

[2] Stefan Funk. *A new population of very high-energy γ-ray source detected with H.E.S.S. in the inner part of the Milky Way.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2005.

[3] Malcolm S. Longair. *The acceleration of high energy particles*, volume 2, page 344–364. Cambridge University Press, 2 edition, 1994.

[4] Roger Blandford and David Eichler. Particle acceleration at astrophysical shocks: A theory of cosmic ray origin. *Phys. Rep.*, 154(1):1–75, October 1987.

[5] Felix A. Aharonian. *Very high energy cosmic gamma radiation : a crucial window on the extreme Universe.* 2004.

[6] F. A. Aharonian, A. M. Atoyan, and T. Kifune. Inverse Compton gamma radiation of faint synchrotron X-ray nebulae around pulsars. *Monthly Notices of the Royal Astronomical Society*, 291(1):162–176, 10 1997.

[7] H.E.S.S. Collaboration and Abdalla, H. et al. The h.e.s.s. galactic plane survey. *A&A*, 612:A1, 2018.

[8] Jörg Frochte. *Maschinelles Lernen, Grundlagen und Algorithmen in Python.* Carl Hanser Verlag München, 2018.

[9] V. Kishore Ayyadevara. *Pro Machine Learning Algorithms.* Apress L.P., 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[13] Constantin Steppa and Kathrin Egberts. Modelling the galactic very-high-energy γ-ray source population. *Astronomy Astrophysics*, 643(A137):11, 2020.

[14] D A Green. Constraints on the distribution of supernova remnants with Galactocentric radius. *Monthly Notices of the Royal Astronomical Society*, 454(2):1517–1524, 10 2015.

[15] Yusifov, I. and Küçük, I. Revisiting the radial distribution of pulsars in the galaxy. *A&A*, 422(2):545–553, 2004.

[16] T. Steiman-Cameron, Mark Wolfire, and and Hollenbach. Cobe and the galactic interstellar medium: Geometry of the spiral arms from fir cooling lines. *The Astrophysical Journal*, 722:1460, 09 2010.

[17] M. Werner, R. Kissmann, A.W. Strong, and O. Reimer. Spiral arms as cosmic ray source distributions. *Astroparticle Physics*, 64:18–33, Apr 2015.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[22] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[25] Martin Riedmiller and Heinrich Braun. Rprop - a fast adaptive learning algorithm. Technical report, Proc. of ISCIS VII), Universitat, 1992.

[26] Stephen P. Reynolds. Supernova remnants at high energy. *Annual Review of Astronomy and Astrophysics*, 46(1):89–126, 2008.

[27] H.E.S.S. Collaboration and Abdalla, H. et al. The population of tev pulsar wind nebulae in the h.e.s.s. galactic plane survey. *A&A*, 612:A2, 2018.

[28] H.E.S.S. Collaboration and Abdalla, H. et al. Population study of galactic supernova remnants at very high energies with h.e.s.s. *A&A*, 612:A3, 2018.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Direkt oder dem Sinn nach übernommene Gedanken aus anderen Werken sind als solche kenntlich gemacht.

Die Arbeit wurde weder einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht.

Potsdam, den 29. März 2021